THE COMBINATION OF A GLOBAL ROUTING ALGORITHM AND A PATH-FINDING ALGORITHM FOR AN UNMANNED ROVING VEHICLE

by

Marvin Hubert Floom, Jr.

United States
Naval Postgraduate School

THESIS

THE COMBINATION OF A GLOBAL ROUTING ALGORITHM AND A PATH-FINDING ALGORITHM FOR AN UNMANNED ROVING VEHICLE

by

Marvin Hubert Floom, Jr.

October 1969

The Combination of a Global Routing Algorithm and a Path-Finding Algorithm for an Unmanned Roving Vehicle

by

Marvin Hubert Floom, Jr.
Second Lieutenant, United States Marine Corps
B.S., United States Naval Academy, 1968

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from

NAVAL POSTGRADUATE SCHOOL
October 1969

ABSTRACT

In future space missions it is planned that an unmanned robot will be sent to explore the other planets' surface. Control of the vehicle from earth is unrealistic because of the long delay time in the transmission of data. From a gross knowledge of the terrain a global routing algorithm can be used to find an optimal path from one point to another. A survey was undertaken to find an algorithm best suited for this use. Dynamic programming was selected and in combination with Lim's path-finding algorithm proved to be successful in simulated vehicle explorations over terrain represented by Gaussian density functions.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF ILLUSTRATIONS

ACKNOWLEDGEMENT

The author would like to express his sincere appreciation for the help, guidance and understanding given by his advisor, Dr. Donald E. Kirk. The author would also like to express his gratitude for the assistance and cooperation given by the staff of the computer facility.

I. INTRODUCTION

With man's first step on the moon it becomes increasingly apparent that future missions will carry him to the other planets in the solar system. It is anticipated that preceding man's exploration of these other planets some kind of unmanned robot will be sent to carry out a preliminary investigation. Because of the long time delays in the transmission of data it will be necessary to have the robot maneuver itself on the surface, using a computer that is onboard or in an orbiting satellite. Prior to placing the robot on the planet, various orbital reconnaissances will be made and a gross knowledge of the topography of the planet will be available.

From this knowledge a landing site can be chosen and the interesting target areas to be visited can be determined. A pathfinding procedure that can be used to maneuver the robot over the terrain and around hazards has been developed (Ref. L-2). This computational procedure, called the local algorithm, uses only onboard sensor information and hence may generate paths which are inefficient in terms of elapsed time, consumed energy, or distance travelled. To compensate for this deficiency, a nominal path can be found by a routing algorithm that uses the knowledge available from photoreconnaissance data (Ref. K-2). The robot follows the nominal path which has been programmed into the

computer. By using local sensory information the robot proceeds along the nominal path until it detects a hazard. The robot then switches to the local path-finding algorithm that uses only local sensory information to proceed around the hazard.

A study of several routing algorithms was undertaken and the results are contained in Chapter II. Chapter III gives the development of the local path-finding algorithm. In order to show the combined operation of the two routing algorithms, a means of simulating suitable terrain was devised -- this is also contained in Chapter III as well as the results of a number of simulations. Combined operation of the global and local algorithms proved to be feasible.

II. ROUTING ALGORITHMS

A. INTRODUCTION

Assume that it is desired to find the shortest path between two cities on a map. For a simple map the solution is trivial. As the number of cities on the map increases, the calculations become tedious. Without knowledge of routing algorithms the task becomes insurmountable.

Routing algorithms have found many uses other than finding the shortest route across a map. The telephone companies have used such algorithms extensively in trunk circuits. If a particular trunk circuit is busy, the circuitry will endeavor to find the next shortest trunk circuit between the cities being called. Industries in the production of consumer commodities make use of routing algorithms in their assembly lines. If a particular section of the assembly is inoperative, or in need of repair, the product is routed to another assembly line where the other components can be put on the article. This makes the most efficient use of the total assembly plant. Once a product is produced it must be shipped. The best and fastest shipping routes are calculated by the use of routing algorithms.

It is intended that the algorithms presented will show just how the optimal paths or procedures can be found. The methods presented require no foresight or ingenuity and

thus deserve to be defined as algorithms. They can easily be used in a machine, either a special purpose or general purpose digital computer.

In the execution of the algorithms, the particular costs, either between cities, or between nodes, or along a trunk circuit, can be arrived at by determining the time to travel between two points, or the total distance travelled. The cost can also include the amount of energy used, or may be a combination of any of these factors.

B. DYNAMIC PROGRAMMING

A very useful type of routing algorithm is based on dynamic programming. Dynamic programming, as developed by R. E. Bellman (Refs. B-1, B-2), is a technique of optimal control as well as a routing algorithm. In formulating any optimal problem a performance measure or cost must be established.

When applying dynamic programming the performance measure may include penalties for not reaching the final point. In the case of routing, the final point must be reached so that the cost is infinite for not reaching the end point. The performance measure can include the amount of time spent or the fuel consumed by the vehicle. These costs may be summed over the entire journey or only parts of it. More importance may be given to the amount of fuel used than to the time spent. In addition, dynamic

programming can be applied to allocation problems. In allocation problems it is desired to maximize the value of the cargo stored in a specified amount of space.

In the case of routing, the cost placed in the performance measure is usually the distance travelled. This distance can be calculated for both two-dimensional and three-dimensional Euclidean surfaces. Costs for time and energy spent in travelling across these surfaces can also be used.

In dynamic programming, the principle of optimality (Refs. B-1, K-1) is used to find the optimal route. For example, if the costs of going from city to city are as shown in Fig. (2.1), the optimal path from Miami to New York goes through Atlanta and Washington. The cost for this route is 19. Because the segment from Washington to New York is part of the optimal path, it is also optimal from Washington to New York. Bellman (Ref. B-3) has called the above property the principle of optimality:

"An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision."

To illustrate this principle suppose that the optimal route from Washington to New York passes through Columbus. This says that the optimal path from Miami to New York passes through Columbus and it is easily seen that this is not the case; the cost from Washington to Columbus to New

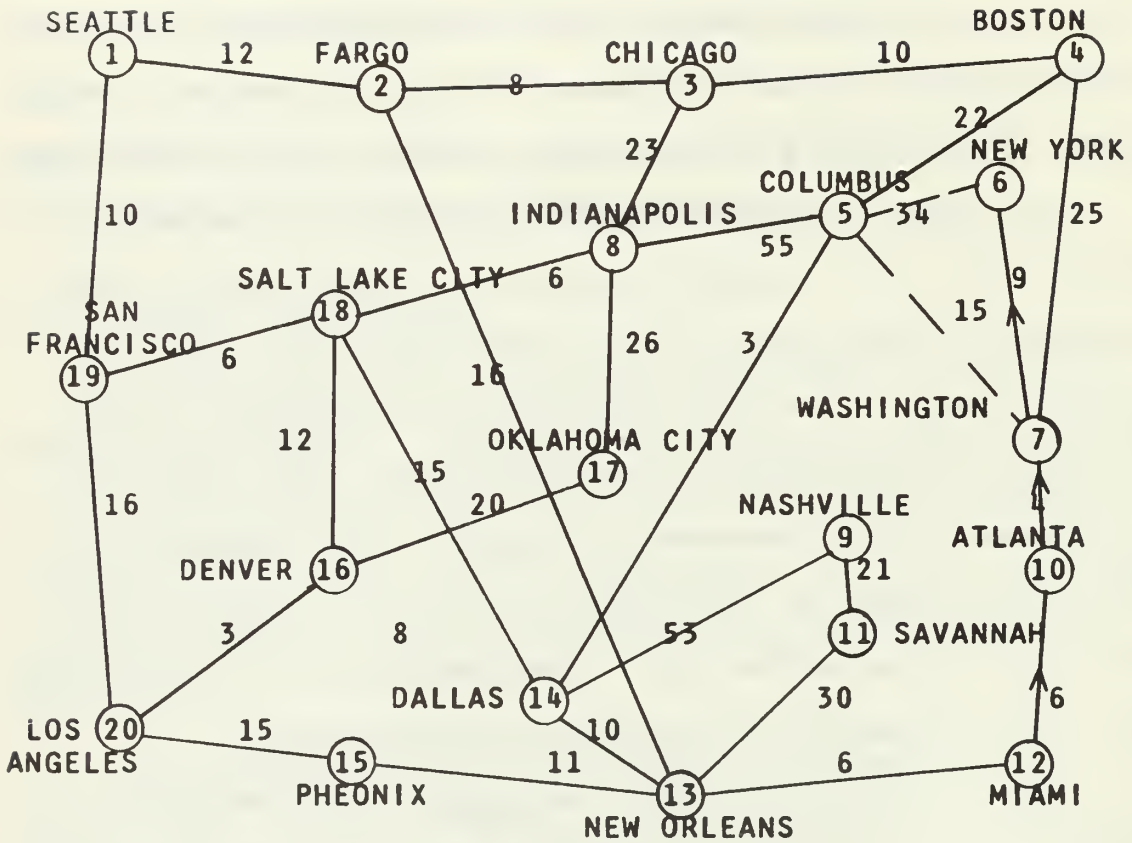


Fig. 2.1 Illustration of the principle of optimality.

York is 49, compared to 9 from Washington to New York. The path from Miami which includes Columbus is longer than the one that goes straight from Washington to New York. This violates the condition that the path including Columbus is the optimal path; therefore, the optimal path goes from Miami through Atlanta and Washington to New York. By the principle of optimality the route from Washington directly to New York is optimal.

Let (X_i, Y_i) $i = 1,2,.....,N$ be the coordinates of a set of N points on a plane. Let the cost of moving from the point L, (X_L, Y_L), to the point M, (X_M, Y_M), along the straight line joining these two points be designated t_{LM}. Assume that t_{LM} has been calculated for all L, M = 1,2,...N and that

$$t_{LM} \begin{cases} > 0 & L \neq M \\ = 0 & L = M \end{cases} \qquad L,M = 1,2,...N \qquad (2.1)$$

The problem is to find a minimum-cost path, specified by a sequence of nodes, that is to be travelled from some initial point L to some specified end point M. This path will be designated the optimal path from L to M.

From the principle of optimality (Refs. B-1, K-1) the functional recurrence equation

$$c_{LM} = \min_{\substack{K \\ K \neq L}} \left\{ t_{LK} + c_{KM} \right\} \quad L,M = 1,2,...N \qquad (2.2)$$

is obtained. c_{LM} is the optimal or minimum cost of moving from node L to M. The number of intermediate points on the optimal path cannot exceed N-2, because this indicates that there is a loop in the path, and by eliminating the loop a smaller cost would result.

The initial conditions for Eq. (2.2) are

$$c_{LL} = 0 , \quad L = 1,2,...N \tag{2.3}$$

Picard's method of successive approximations can be used to solve Eq. (2.2); this approach leads to the equation

$$c_{LM}^{j+1} = \min_{\substack{K \\ K \neq L}} \left\{ t_{LK} + c_{KM}^{j} \right\} \quad L,M = 1,2,...N \tag{2.4}$$

with the initial value

$$c_{LM}^{o} = t_{LM} \quad L,M = 1,2,...N. \tag{2.5}$$

The successive approximations have the following interpretation:

1. c_{LM}^{o} is the minimum cost to go from L to M via no intermediate nodes;

2.

$$c_{LM}^{1} = \min_{\substack{K \\ K \neq L}} \left\{ t_{LK} + c_{KM}^{o} \right\} \tag{2.6}$$

or

$$= \min_{\substack{K \\ K \neq L}} \left\{ t_{LK} + t_{KM} \right\}$$

is the minimum cost to go from L to M via <u>at most</u> one intermediate node;

3.

$$c_{LM}^{j+1} = \min_{\substack{K \\ K \neq L}} \left\{ t_{LK} + c_{KM}^{o} \right\} \tag{2.7}$$

is the minimum cost to go from L to M via at most j+1 intermediate nodes.

To solve for c_{LM}^{j+1}, t_{LK}, $K \neq L$, $K = 1,2,\ldots N$ (the Lth row of the $\underset{\sim}{T}$ matrix) and c_{KM}^{j} $K \neq M$, $K = 1,2,\ldots N$ (the Mth column of the $\underset{\sim}{C}$ matrix) are needed. The solution is obtained by comparing the values $t_{LK} + c_{KM}^{j}$ for $K = 1,2,\ldots N$, $K \neq M$, to find the minimum value of K.

After the minimizing value of K is calculated, the cost associated with this K is stored as the LM element of the $\underset{\sim}{C}$ matrix. The dimensions of the matrix are depedent on the number of nodes; therefore, the $\underset{\sim}{C}$ matrix is an N by N square matrix.

After finding the minimizing value of K, this value is also stored in the LM position of the node matrix $\underset{\sim}{N}$. Once the iterative process has terminated, the node matrix can be used to find the intermediate nodes along the optimal path from L to M.

The use of this algorithm is best seen by an example. On the map in Fig. (2.2) the costs between cities are shown. The costs are not the actual distances, but are arbitrary numbers picked at random for this example. In some

cases there are no direct paths between cities. Where there is no direct connection, the cost between the two cities was set at 500. This cost is large enough to prevent the algorithm from selecting the direct path between the two cities. In Table II.1a the $\underset{\sim}{T}$ matrix was solved for. Due to the large number of nodes only a section of the map was used in this example. Table II.1b shows the section of the $\underset{\sim}{T}$ matrix that was used. Because only a section of the total T matrix was used, the optimal paths calculated in the example may differ from those calculated by using the entire $\underset{\sim}{T}$ matrix.

Set $C^o = T$ for the successive approximation Eq. (2.4). To solve for the first row of the $\underset{\sim}{C}^1$ matrix, the first calculation is:

$$c_{33}^1 = 0$$

$$c_{34}^1 = \min \{t_{34} + c_{44}^o,\ t_{35} + c_{35}^o,\ t_{36} + c_{64}^o,$$

$$t_{37} + c_{74}^o,\ t_{38} + c_{84}^o\}$$

$$= \min \{10 + 0,\ 500 + 22{,}500 + 500,$$

$$500 + 25{,}23 + 500\} = 10$$

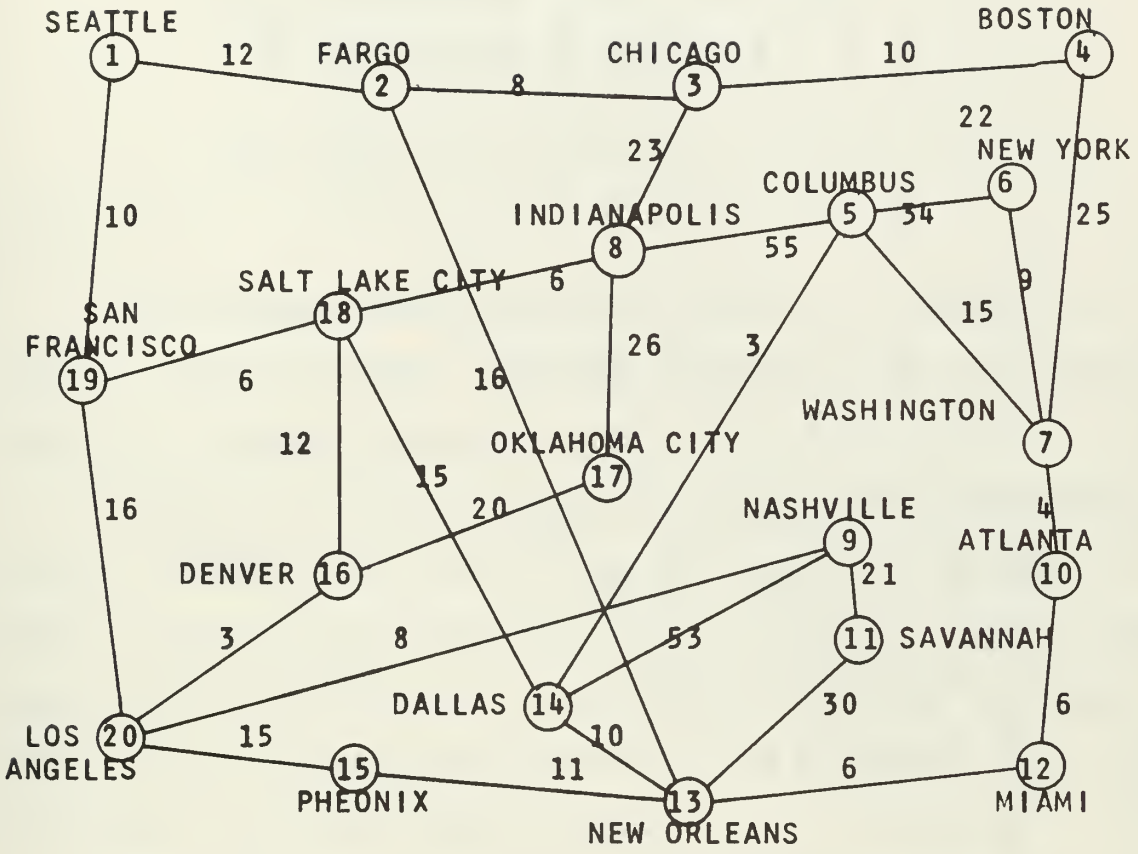


Fig. 2.2 The costs between cities.

Table II.1a The Entire Cost Matrix for Fig. 2.2

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	SEATTLE	0	12	500	500	500	500	500	500	500	500	500	500	500	500	500	500	500	500	10	500
2	FARGO		0	8	500	500	500	500	500	500	500	500	500	16	500	500	500	500	500	500	500
3	CHICAGO			0	10	500	500	500	23	500	500	500	500	500	500	500	500	500	500	500	500
4	BOSTON				0	22	500	25	500	500	500	500	500	500	500	500	500	500	500	500	500
5	COLUMBUS					0	34	15	55	500	500	500	500	500	3	500	500	500	500	500	500
6	NEW YORK						0	9	500	500	500	500	500	500	500	500	500	500	500	500	500
7	WASHINGTON							0	500	500	4	500	500	500	500	500	500	500	500	500	500
8	INDIANAPOLIS								0	500	500	500	500	500	500	500	500	26	6	500	500
9	NASHVILLE									0	500	21	500	500	53	500	500	500	500	500	8
10	ATLANTA										0	500	6	500	500	500	500	500	500	500	500
11	SAVANNAH											0	500	30	500	500	500	500	500	500	500
12	MIAMI												0	6	500	500	500	500	500	500	500
13	NEW ORLEANS													0	10	11	500	500	500	500	500
14	DALLAS														0	500	500	500	15	500	500
15	PHOENIX															0	500	500	500	500	15
16	DENVER																0	20	500	500	3
17	OKLAHOMA CITY																	0	500	500	500
18	SALT LAKE CITY																		0	6	500
19	SAN FRANCISCO																			0	16
20	LOS ANGELES																				0

and the n_{34} element of the node matrix is $n_{34} = 4$. The next step gives;

$$c_{35}^{1} = \min \{t_{34} + c_{45}^{o},\ t_{35} + c_{55}^{o},$$
$$t_{36} + c_{65}^{o},\ t_{37} + c_{75}^{o},\ t_{38} + c_{85}^{o}\}$$
$$= \min \{10 + 22{,}500 + 0{,}500 + 34,$$
$$500 + 15{,}23 + 55\} = 32$$

and $n_{35} = 4$.

This process continues until all of the elements in the row are calculated; then the remaining rows are calculated in a similar manner.

A second iteration was carried out in the manner prescribed. Since the $\underset{\sim}{C}^{1}$ and $\underset{\sim}{C}^{2}$ matrices were not the same (Table II.2), a third iteration was carried out. After the third iteration there were no changes in the cost and node matrices, so the solution yielded $\underset{\sim}{C}^{2}$ and $\underset{\sim}{N}^{2}$ (Table II.3) as the optimal cost and node matrices.

In order to see how to use the optimal node matrix, suppose that it is desired to travel from New York (6) to Indianapolis (8). Looking up n_{68} (see Table II.3) it is found that 7 is the first intermediate point. By looking up the 7,8 element in the node matrix, 4 is found to be the first intermediate node on the optimal path from 7 to 8, and the second intermediate node on the optimal path from 6 to 8. The 4,8 element is 3 and this becomes the third intermediate node. The 3,8 element of the node matrix is

		3	4	5	6	7	8
3	CHICAGO	0	10	500	500	500	23
4	BOSTON	10	0	22	500	25	500
5	COLUMBUS	500	22	0	34	15	55
6	NEW YORK	500	500	34	0	9	500
7	WASHINGTON	500	25	15	9	0	500
8	INDIANAPOLIS	23	500	55	500	500	0

Table II.1b

The Cost Matrix for the Dynamic Programming Example

found to be the final point 8. By the principle of optimality n_{68} includes 7, 4, and 3; therefore the optimal path is 6-7-4-3-8.

The dynamic programming approach requires storage for two matrices whose dimensions depend on the number of points or nodes. This may mean a great deal of storage for a single problem, but this is compensated for by the small number of calculations.

C. DANTZIG'S SHORTEST-ROUTE ALGORITHM

George B. Dantzig (Ref. D-1) has applied linear programming methods for the solution of discrete-variable extremum problems to obtain two routing algorithms. Dantzig's algorithms have been used in routing telephone calls, in circuit analysis and in shipping, but because of the many calculations and large storage requirements have proven to be of limited value.

As in dynamic programming, the costs are arrived at by considering the time taken, or the distance travelled, or the energy used to get from one point to another. For example, using the same map as in the dynamic programming example (Fig. 2.2), assume that it is desired to transfer a package from Los Angeles to Boston, and that there is no direct route. In this case the package must be shipped along the lines of the road network until it arrives at Boston.

The Cost Matrix

$C^1 =$

		3	4	5	6	7	8
3	CHICAGO	0	10	32	500	35	23
4	BOSTON	10	0	22	34	25	33
5	COLUMBUS	32	22	0	24	15	55
6	NEW YORK	500	34	24	0	9	89
7	WASHINGTON	35	25	15	9	0	70
8	INDIANAPOLIS	23	33	55	89	70	0

The Node Matrix

$N^1 =$

		3	4	5	6	7	8
3	COLUMBUS	3	4	3	6	4	8
4	BOSTON	3	4	4	7	7	3
5	COLUMBUS	4	4	5	7	7	8
6	NEW YORK	3	7	5	6	7	5
7	WASHINGTON	4	4	5	6	7	5
8	INDIANAPOLIS	3	3	5	5	5	8

Table II.2

The Cost and Node Matrices after the First Iteration

The Cost Matrix

$\underset{\sim}{C}^2 =$

		3	4	5	6	7	8
3	CHICAGO	0	10	32	44	35	23
4	BOSTON	10	0	22	34	25	33
5	COLUMBUS	32	22	0	24	15	55
6	NEW YORK	44	34	24	0	9	67
7	WASHINGTON	35	25	15	9	0	58
8	INDIANAPOLIS	23	33	55	67	58	0

The Node Matrix

$\underset{\sim}{N}^2 =$

		3	4	5	6	7	8
3	CHICAGO	3	4	4	4	4	8
4	BOSTON	3	4	5	7	7	3
5	COLUMBUS	4	4	5	7	7	8
6	NEW YORK	7	7	5	6	7	7
7	WASHINGTON	4	4	5	6	7	4
8	INDIANAPOLIS	3	3	5	3	3	8

Table II.3

The Cost and Node Matrices after the Second Iteration

For $i \neq j$, let $x_{ij} = 1$ mean that the package is sent directly from city i to city j and let $x_{ij} = 0$ mean that it is not. x_{ii} is 1 if the package comes to city i from some adjacent city and 0 if the package does not. If a route exists between two cities these cities are said to be connected. Under these conditions a system of constraints (Table II.4) can be written.

For example, the fourth equation in (2.8) indicates that the amount shipped into Boston is equal to unity. The last equation in (2.9) states that the amount shipped out of Los Angeles is equal to unity.

Subject to the constraints in Table II.4 and the further constraint $x_{ij} = 0$ or 1 for $i \neq j$, the form

$$\sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} x_{ij} = z \qquad n = 20 \tag{2.10}$$

is minimized -- d_{ij} is the cost of travelling between city i and city j.

The x_{ij}'s in the constraints are unknowns and hence all possibilities and combinations of these equations must be tried. Some of the calculations are presented here:

$$\begin{aligned} z = \{ & d_{11}x_{11} + d_{12}x_{12} + d_{13}x_{13} \cdots d_{1,20}x_{1,20} \\ & + d_{21}x_{21} + d_{22}x_{22} + d_{23}x_{23} \cdots d_{21,20}x_{2,20} \\ & + \cdots\cdots \\ & + d_{19,1}x_{19,1} + d_{19,2}x_{19,2} + d_{19,3}x_{19,3} \cdots d_{19,20}x_{19,20} \\ & + d_{11}x_{11} + d_{21}x_{21} + d_{31}x_{31} \cdots d_{20,1}x_{20,1} \\ & + \cdots\cdots \\ & + d_{1,20}x_{1,20} + d_{2,20}x_{2,20} + d_{3,20}x_{3,20} \cdots d_{20,20}x_{20,20} \} \end{aligned}$$

$$
\begin{array}{llllll}
x_{21} & & & +x_{19,1} & =x_{11} & \\
x_{12} & +x_{32} & +x_{13,2} & & =x_{22} & \\
x_{23} & +x_{43} & +x_{17,3} & & =x_{33} & \\
x_{34} & +x_{54} & +x_{74} & & =1 & \\
x_{45} & +x_{65} & +x_{85}+x_{75} & +x_{14,5} & =x_{55} & (2.8) \\
x_{56} & +x_{76} & & & =x_{66} & \\
\cdots & \cdots & \cdots & \cdots & \cdots & \\
x_{1,19} & & & +x_{18,19} & =x_{19,19} &
\end{array}
$$

$$
\begin{array}{llllll}
x_{12} & & & +x_{1,19} & =x_{11} & \\
x_{21}+x_{23} & & +x_{2,13} & & =x_{22} & \\
x_{32} & +x_{34} & & +x_{3,12} & =x_{33} & \\
x_{54} & +x_{56}+x_{57}+x_{58} & & +x_{5,14} & =x_{55} & (2.9) \\
\cdots & \cdots & \cdots & \cdots & \cdots & \\
x_{19,1} & & +x_{19,18} & & =x_{19,19} & \\
x_{20,9} & +x_{20,15}+x_{20,16}+x_{20,19} & & & =1 &
\end{array}
$$

Table II.4

The Constraints of the Map in Fig. 2.2

After trying all possibilities of x_{ij} in Table (II.4), the optimal case was found and is shown below.

$x_{44} = 1,\ x_{43} = 1,\ x_{33} = 1,\ x_{38} = 1,\ x_{38} = 1,\ x_{88} = 1,$

$x_{8,18} = 1,\ x_{18,18} = 1,\ x_{18,16} = 1,\ x_{16,16} = 1,\ x_{16,20} = 1,$

$x_{16,18} = 1,\ x_{20,20} = 1,\ x_{34} = 1,\ x_{83} = 1,\ x_{18,8} = 1,$

$x_{20,10} = 1$ and

all of the other x_{ij}'s and x_{ii}'s are equal to zero.

$$z = \{d_{44}x_{44} + d_{43}x_{43} + d_{34}x_{34} + d_{33}x_{33} + d_{38}x_{38}$$

$$+d_{83}x_{83} + d_{88}x_{88} + d_{8,18}x_{8,18} + d_{18,8}x_{18,8}$$

$$+d_{18,18}x_{18,18} + d_{18,16}x_{18,16} + d_{16,18}x_{16,18} + d_{16,16}x_{16,16}$$

$$d_{16,20}x_{16,20} + d_{20,20}x_{20,20}\}$$

$$= (0x1 + 10x1 + 10x1 + 0x1 + 23x1 + 23 \times 1$$

$$0x1 + 6x1 + 6x1 + 0x1 + 12x1 + 12x1 + 0x1$$

$$3x1 + 3x1 + 0x1)$$

$$= 108.$$

After considering that all possibilities of x_{ij} must be tried, it is seen that this method appears to be only as good as direct enumeration. Direct enumeration also requires that all possible paths be tried, and in Dantzig's algorithm this is done by substituting the appropriate

values of x_{ij} into Eq. (2.10). It is concluded that the time spent and amount of storage used in Dantzig's algorithm is excessive.

D. DANTZIG'S GRAPHICAL ROUTING ALGORITHM

In the same article (Ref. D-1), Dantzig presents a graphical approach to the algorithm already explained. By using graphics he makes the problem easier to visualize, but in no way shortens the process. All possible paths must be tried in order to be sure that the optimal path is found.

First a path is chosen from the starting point, Los Angeles, to all of the other cities on the map (see Fig. 2.3). These paths, indicated by arrows, are completely arbitrary. The routes associated with the arrows cannot form loops. First the distance from Los Angeles to city i along the path indicated by the arrows is calculated. This distance, or cost, is indicated by t_i . For example, from Los Angeles to Chicago the distance along the route shown is 38 plus 8; hence $t_3 = 46$. The next step is to find the optimal route from Los Angeles to all other cities. To do this, the numbers in the circles have to be justified as the shortest distances. To test whether they are minimum for each directed link of the network joining city i to city j, the distance d_{ij} is compared with $t_i - t_j$. If $t_i - t_j \leq d_{ij}$ for each of the links, the optimal path from Los Angeles to any other city has been found.

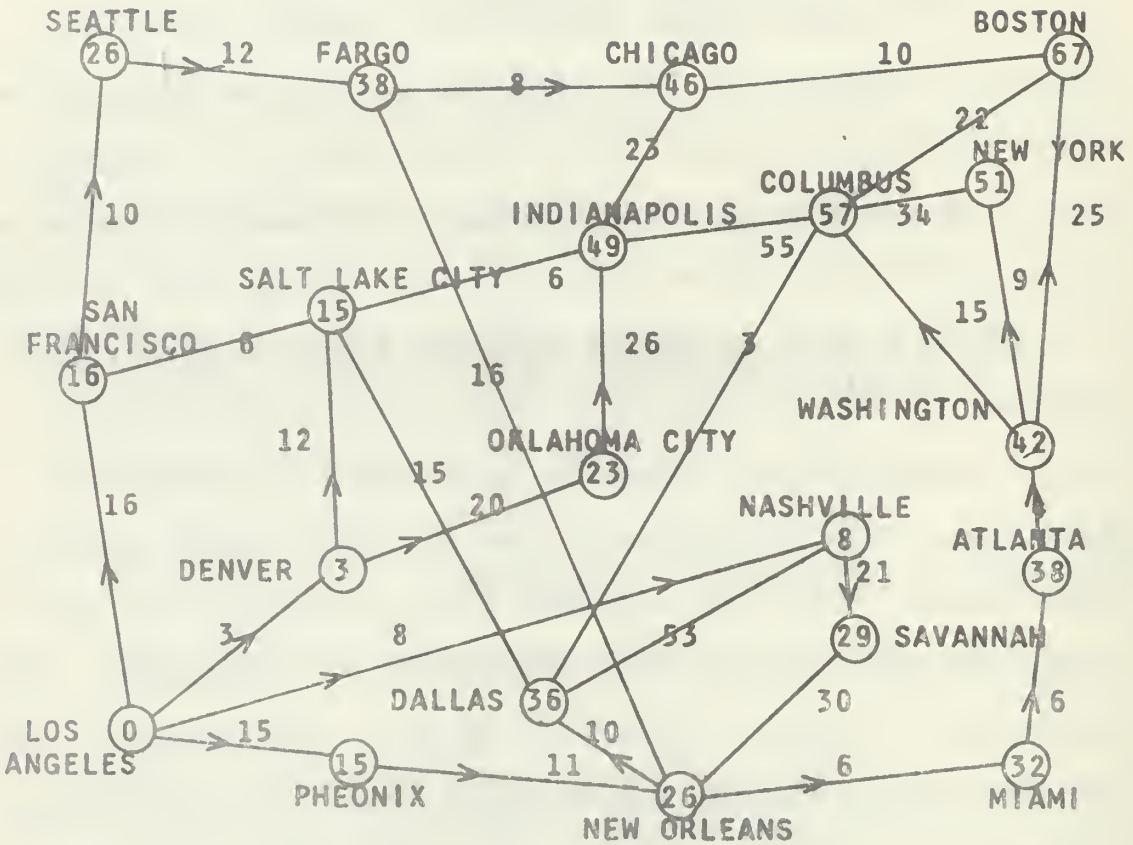


Fig. 2.3 Illustration of Dantzig's graphical algorithm.

For example, since the distance d_{34} from Chicago to Boston is 10 compared to $t_3-t_4 = 67-46 = 21$, the arrow is placed between Chicago and Boston instead of between Washington and Boston, and t_{34} is changed from 67 to 56. All other possible checks are made and the shortest route from Los Angeles to Boston includes Denver, Salt Lake City, Indianapolis and Chicago (see Fig. 2.4).

Dantzig's graphical routing algorithm is much easier to understand, but it still has the disadvantage of being a trial-and-error method. It would not be advantageous to program this algorithm for computer use, because the number of calculations and logic statements is excessive when compared to dynamic programming. The number of steps in the calculation is dependent on the number of cities and how they are connected to one another. The number of steps equals $\sum_{i=1}^{n} m_i$ (n = 20 in this example) where m_i is the number of connections to city i. The number of steps in this algorithm is greater than the number required by dynamic programming.

E. MOORE'S SHORTEST PATH THROUGH A MAZE

By taking a different approach to the problem of shortest routes, Moore (Ref. M-1) has obtained some interesting results that have many practical applications. Among these applications are the routing of toll calls through trunk circuits, and the determination of shipping and transportation routes.

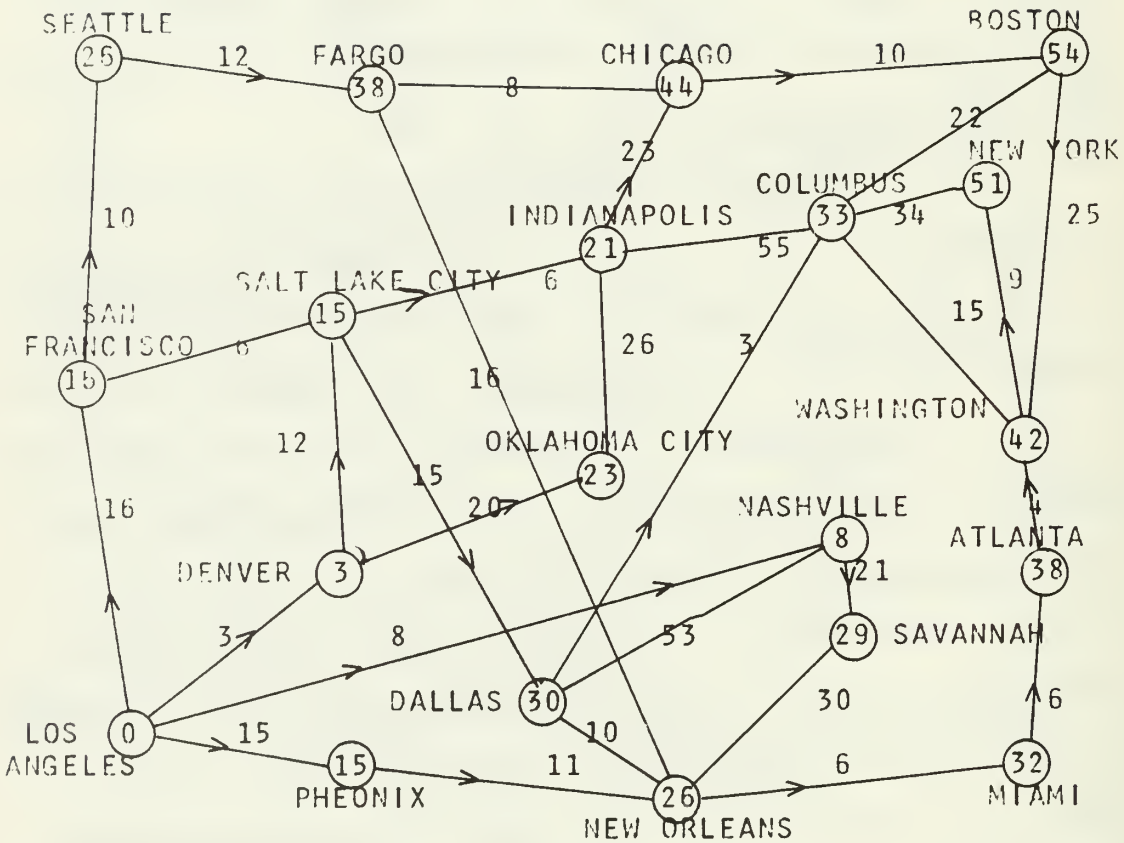


Fig. 2.4 The solution to the example of Dantzig's graphical algorithm.

1. Algorithm A

In his first algorithm, Moore divides all the nodes, or cities, into a unity-cost space; the costs between each of the cities are the same, or unity, in value.

Suppose that it is desired to find a path with the minimum number of intermediate cities between Los Angeles and Boston (Fig. 2.5). In order to carry out the algorithm, the number zero first is placed in the circle adjoining Los Angeles (see Fig. 2.6). In the second step, the number one is written on the cities connected to Los Angeles - San Francisco, Denver, Nashville, and Phoenix. In the next step, the number two is written in the cities adjoining those obtained in the previous step; thus Seattle, Salt Lake City, Oklahoma City, Dallas, Savannah and New Orleans are designated with a two. In the fourth step, as in the previous two steps, the number three is written on the cities connected to those in the third step. The adjoining cities are Fargo, Indianapolis, Columbus and Miami. In the fifth step the number four is placed in all of the cities adjoining those in step four. In this step the destination, Boston, is reached because it is connected to Columbus. Now the path must be retraced back to the starting point. The sixth, seventh, eighth, and ninth steps simply involve putting an arrow from Boston to Columbus, from Columbus to Dallas, from Dallas to Nashville and from Nashville to Los Angeles. The tenth step involves erasing all of the numbers not on the path itself.

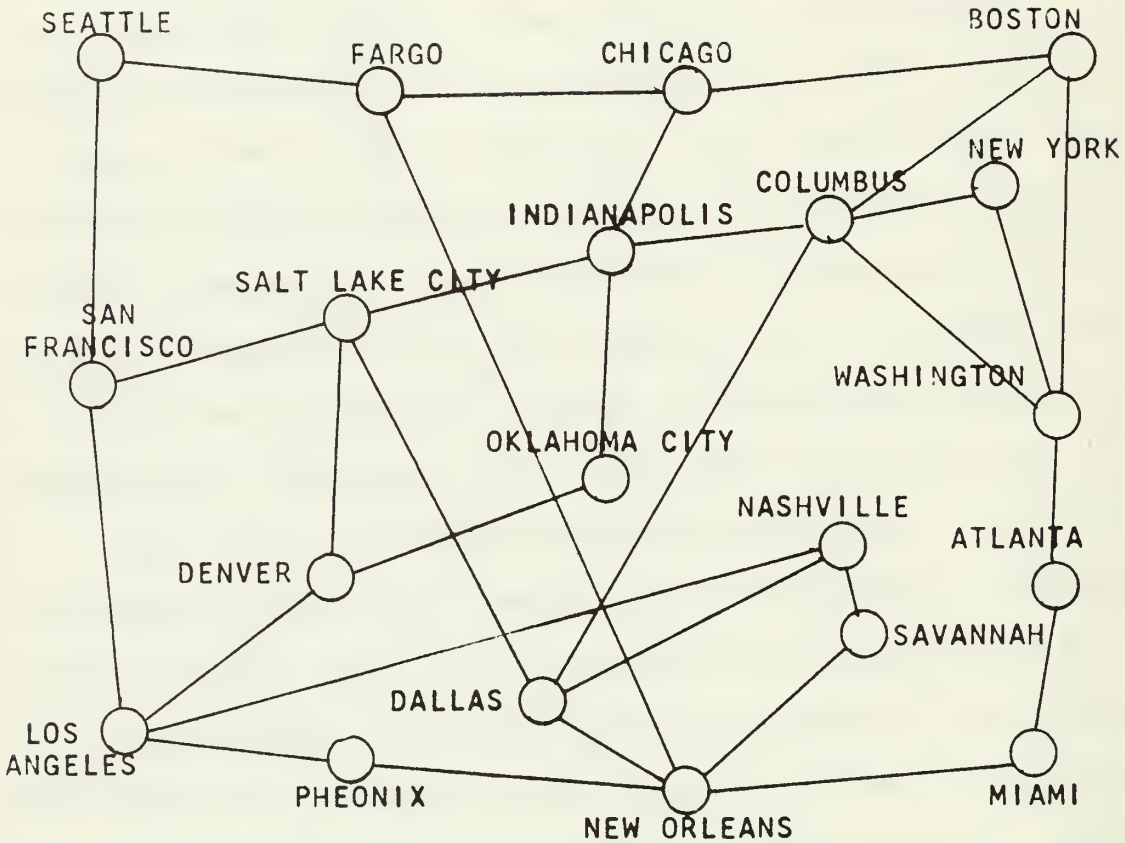


Fig. 2.5 The network connecting several cities.

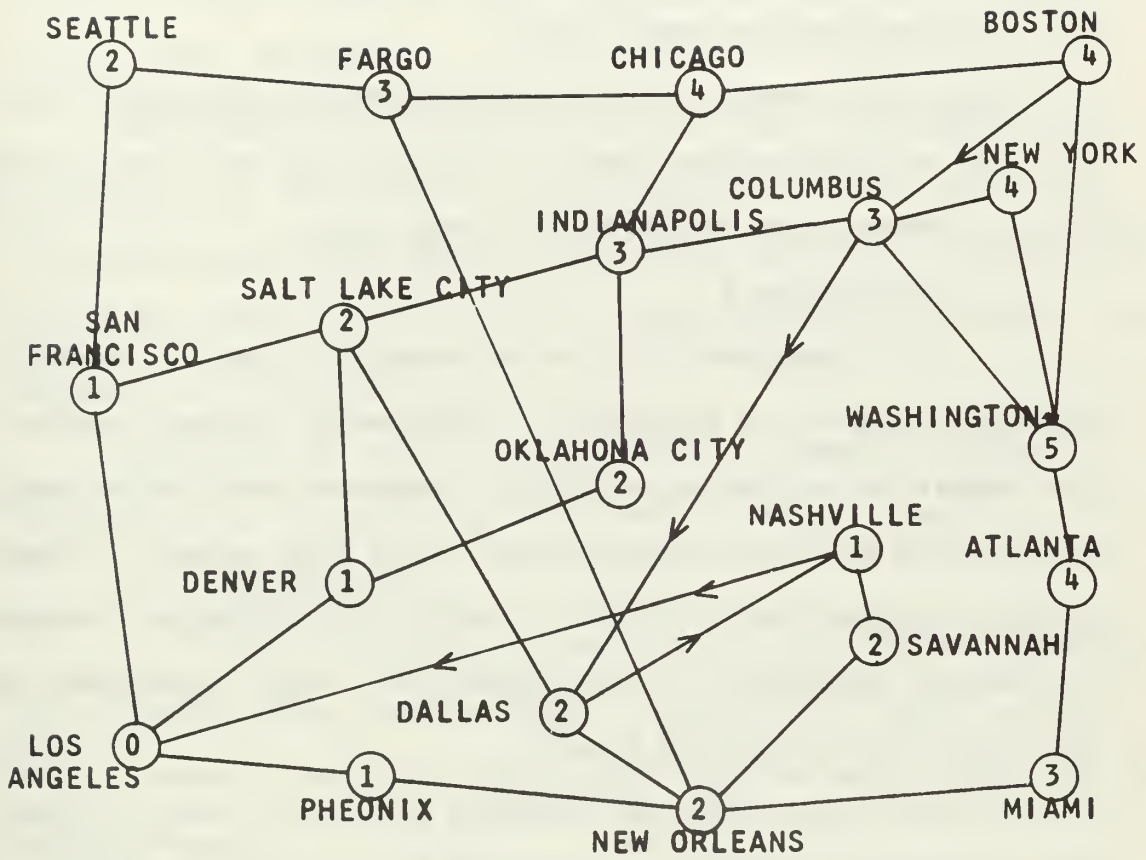


Fig. 2.6 The results of applying algorithm A to Fig. 2.5.

In general, if the length of the shortest path is n it takes 2n+2 steps if this algorithm - which will be referred to as algorithm A - is used. On a digital computer the steps carried out above would be done serially with a few calculations for each city.

The total amount of memory required is dependent on the cost of the optimal path. If n is the cost then $1+\log_2 n$ bits of memory are required for each city.

2. Algorithm B

In algorithm A it is necessary to have a memory capacity capable of handling an arbitrary integer (up to the number of cities on a path). Suppose that it is desired to carry out the same problem using less memory. Moore suggests algorithm B, which by modulo 3 arithmetic, reduces the amount of memory for each node to 2 bits regardless of the number of cities on a path.

At each city only the numbers 0,1, or 2 are written; the appropriate number is obtained by dividing the path length by 3 and writing the remainder in the circle for each city. This number can also be derived by counting 0,1,2,0,1,2, etc., from the starting point until the final destination is reached. As the number of cities increases, it becomes desirable to use a modulo arithmetic with a base of greater magnitude. For any problem, the base of the modulo arithmetic can be adjusted to any number, n, as long as $n \geq 3$. The use of the integers of mod n depends on the

fact that, if the integers are calculated for any set of cities, each city must have an integer label which is equal to one less than or one more than the adjacent label. Integers of mod 3 afford this capability, whereas those of mod 1 and 2 do not.

Algorithm B is carried out with the same number of steps and in the same way as algorithm A. The only difference between the two is that in algorithm B integers of mod 3 are used instead of the normal counting numbers. An example is shown in Fig. (2.7).

Algorithm B reduces the amount of computer memory necessary to solve the problem and takes no longer.

3. Algorithm C

There is still another algorithm, called Algorithm C, to solve this same problem. Algorithm C uses only one bit of memory for each city, but requires more steps in its calculation.

To use this algorithm, zeros are written in all the cities. A one is then written in the first city. In the next step, ones are written in all of the cities that are connected to this first city. In the next step, this same process is repeated. After n+1 steps-- n is the number of cities passed through -- the next group of ones requires that a one be placed in the destination. With a one on the destination, an arrow is drawn to indicate the path taken from the final point back to the city which the final point connects. Then all of the ones are erased.

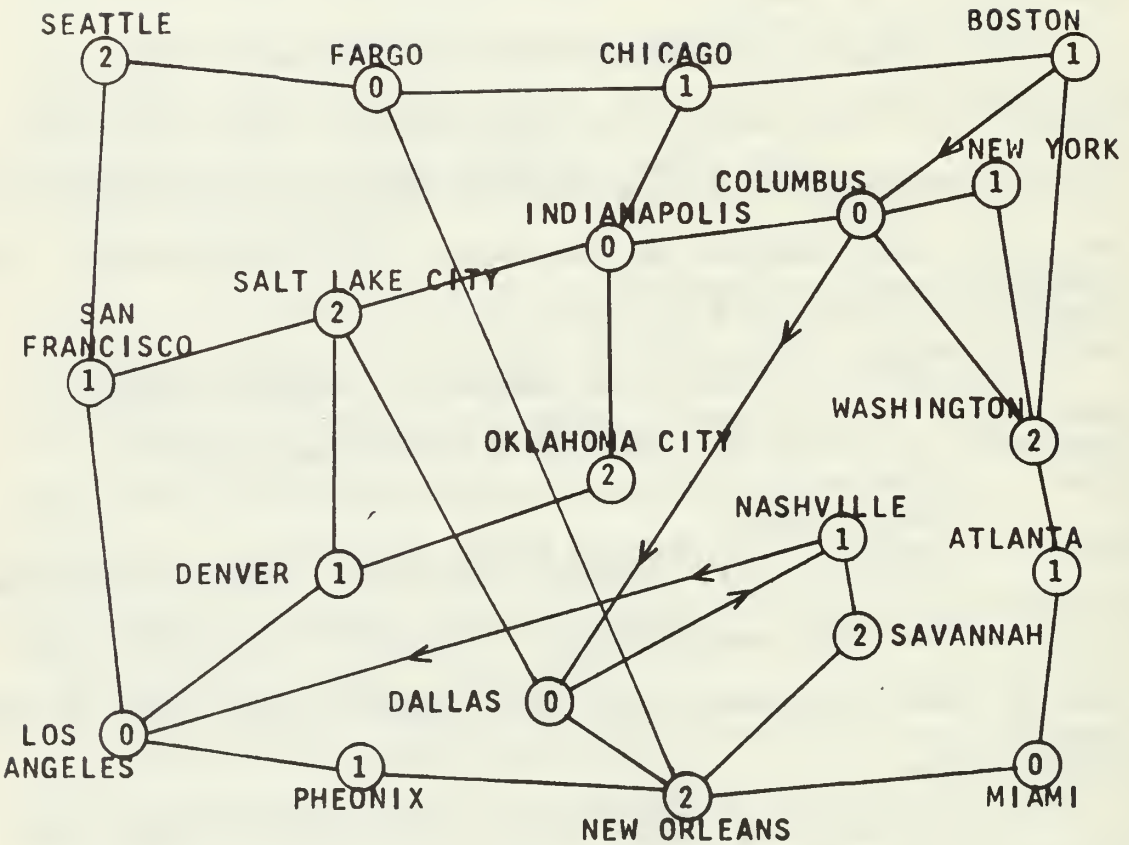


Fig. 2.7 The results of applying algorithm B to Fig. 2.5.

In the next iteration the process is carried out in the same manner but with the final point changed. The final point for the new iteration is the city that connects the final point in the preceding iteration. This process is continued until the arrows reach the starting point.

The following example illustrates the operation of the algorithm. Using the same map as in Fig. (2.5), suppose that it is desired to find the shortest route from Los Angeles to Boston. In the first step a one is placed in the circle for Los Angeles. The second step requires that a one be placed in the cities adjacent to Los Angeles--San Francisco, Denver, Nashville, and Phoenix. In the third step, a one is placed in the cities connected to San Francisco, Denver, Nashville, and Phoenix; these cities are Seattle, Salt Lake City, Oklahoma City, Savannah, Dallas and New Orleans. In the fourth step, ones are placed at Fargo, Indianapolis, Columbus and Miami. In the (n+1)st, or fifth step, a one is placed at Boston and an arrow is drawn from Boston to Columbus (Fig. 2.8). All of the ones are then erased and the process is repeated with Columbus as the final point. After four steps, a one is placed at Columbus and an arrow is drawn from Columbus to Dallas--which becomes the new final point for the next iteration. All of the ones are then erased and the process started over. After (n+1)+(n)+(n-1), or 12 steps, a one is placed at Dallas and an arrow is drawn from Dallas to Nashville (Fig. 2.9). All of the ones are replaced with zeros and a

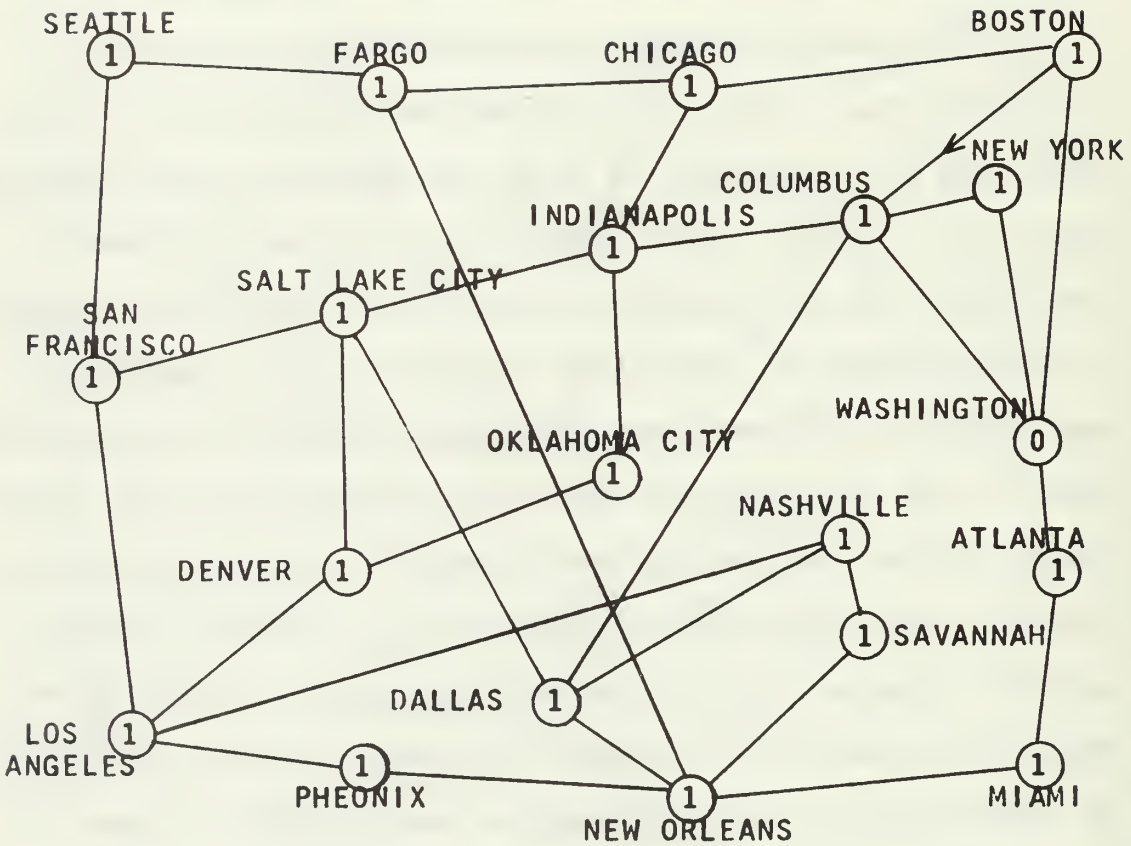


Fig. 2.8 The fifth step in algorithm C.

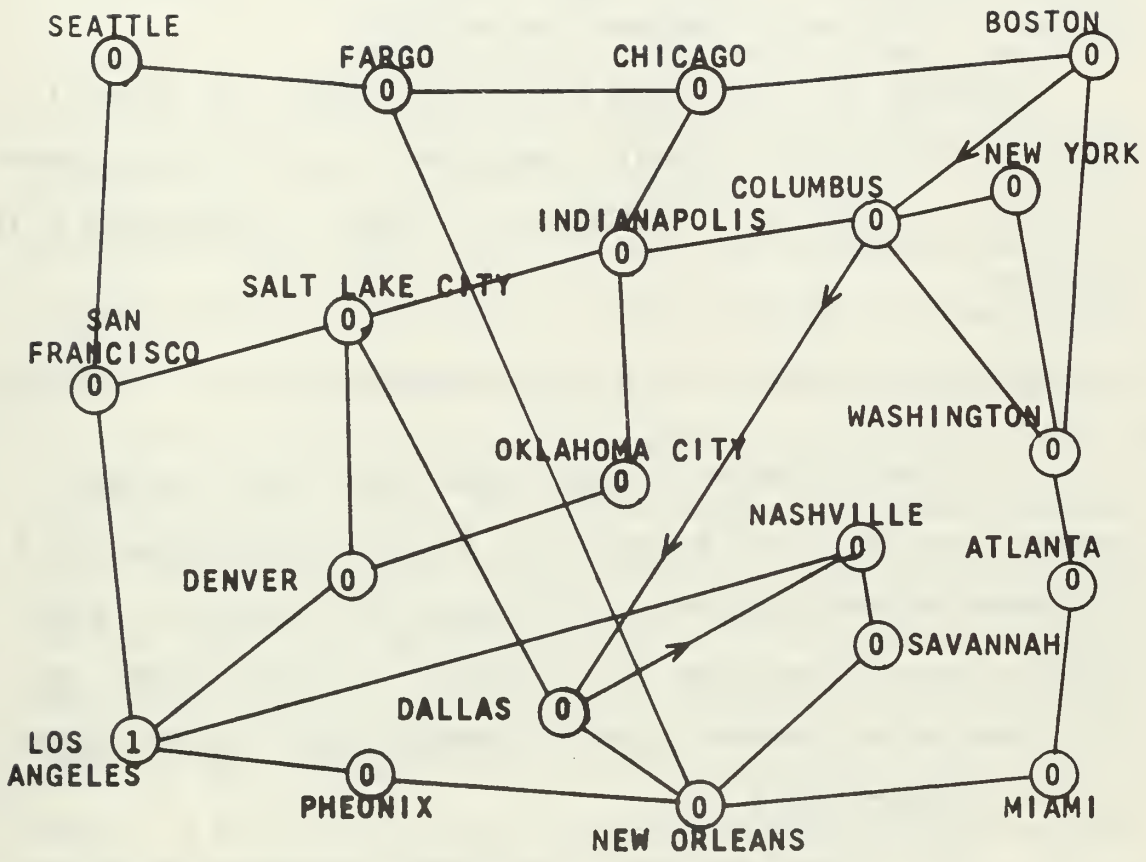


Fig. 2.9 The twelfth step in algorithm C.

one is placed at Los Angeles. A one is placed at Nashville and an arrow is drawn from Nashville to Los Angeles, because Nashville is the new final point. The optimal path is then from Los Angeles to Nashville, to Dallas, to Columbus, to Boston -- the same as determined earlier.

Although the algorithm uses less memory, it takes 15 steps to solve this example, compared to 10 for algorithms A and B. In general, the number of steps in algorithm C is equal to

$$(n+1) + n + (n-1) + \cdots 1 = \tfrac{1}{2}(n+1)(n+2) \qquad (2.11)$$

All three of Moore's algorithms take less time and memory than Dantzig's algorithms; Moore's algorithms are a much more direct approach for finding the solution. With Moore's three algorithms it is easy to find the right one if a compromise between time and memory space is necessary.

4. Algorithm D

The three previous algorithms presented by Moore are all designed to solve the same problem. Because of the unity cost between two adjacent cities, algorithms A, B, and C are too restrictive for the vehicle-routing application. In communications and transportation, the least time elapsed, or the least distance traversed, or the least fuel consumed between two points is often what the problem requires. In Moore's fourth algorithm, he develops a procedure which is capable of handling the different costs

between cities. In this algorithm, which has been labelled algorithm D, it is assumed that the path with the least cost is to be found.

In the first step of algorithm D, the number zero is written on the starting city. The second step consists of writing the costs from the starting city on all of the cities that are connected to starting city. In the third step, the cost from the starting city is written on all the cities that are connected to those cities that had costs written on them in the second step. This cost is arrived at by adding the cost between the latter two cities to the cost placed on the city in the second step. In some cases, a city may be connected to the starting city and also to one of the cities in the second step. If this occurs, the least cost that has been calculated is written on the new city. These steps continue until all of the cities have a cost on them. A path is retraced from the terminal point to the preceding city by checking the costs at all of the adjacent cities. The difference between the costs of the adjacent city and the terminal city is calculated. The difference that equals the cost of the path connecting the two cities, indicates the optimal path between the two cities. An arrow is placed pointing to the adjacent city determined in this manner. This process continues until the arrow points toward the starting city.

For example, in Fig. (2.10) it is desired to get from Los Angeles to Boston with the least cost. In the first step, a zero is placed at Los Angeles. The second step consists of putting the costs of 16 on San Francisco, 3 on Denver, 8 on Nashville and 15 on Phoenix. In the third step, the cost associated with the cities that are adjacent to the cities processed in the preceding step are calculated. Seattle and Salt Lake City are adjacent to San Francisco and the costs to these cities from Los Angeles are 26 and 22 respectively. From Denver to Salt Lake City the cost is only 15; therefore, the cost from Los Angeles to Salt Lake City via Denver is placed at Salt Lake City, because it is less than the cost from Los Angeles to Salt Lake City via San Francisco. The cost to Oklahoma City from Los Angeles via Denver is 23. The cost from Los Angeles to Savannah via Nashville is 29; to Dallas the cost is 61. The cost from Los Angeles to New Orleans via Phoenix is 26. This process is continued until all of the cities have the appropriate costs written on them.

In order to select the path from Los Angeles, it is necessary first to find the difference in cost between Boston and the adjacent cities. By finding the difference that is equal to the cost along that path, an arrow can be placed that indicates the direction to Los Angeles. The difference in cost between Boston and Chicago is 10 -- this is equal to the cost along the direct path from Boston to

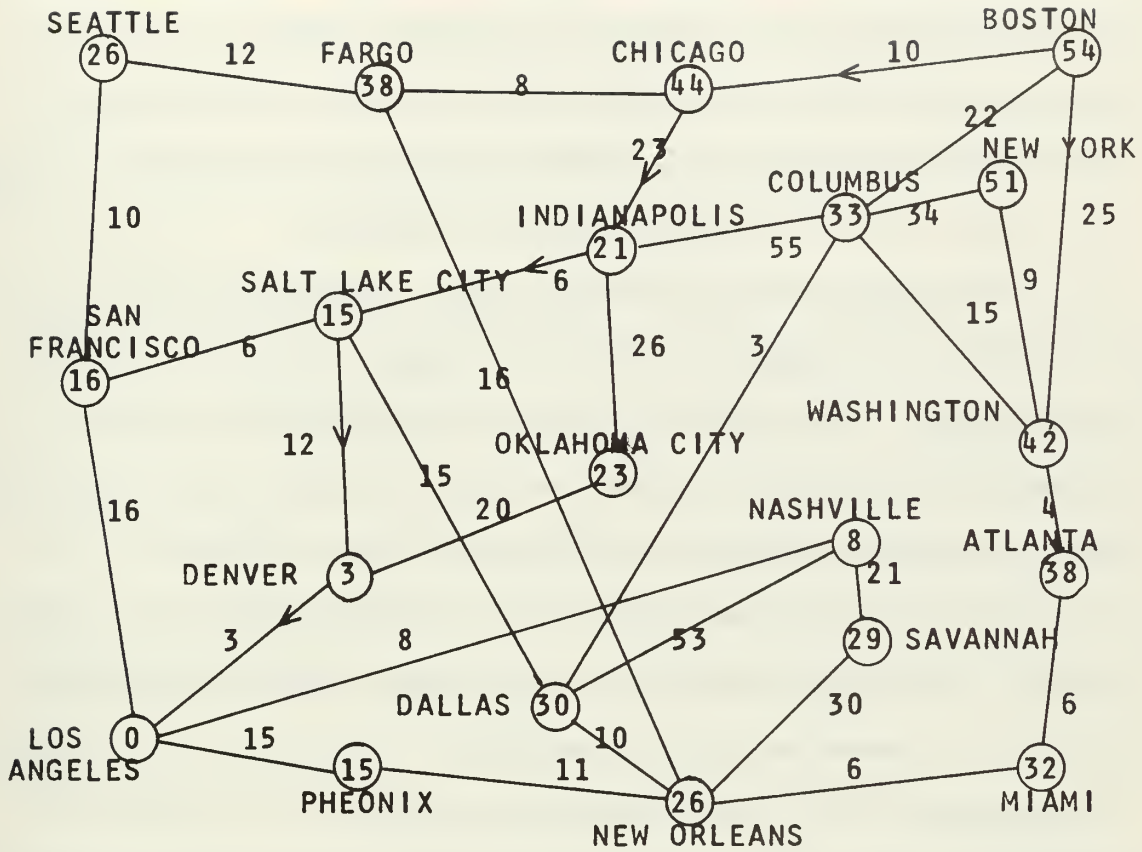


Fig. 2.10 The results of applying algorithm D to Fig. 2.5.

Chicago; therefore, an arrow is then placed pointing toward Chicago. The difference in cost between Chicago and Indianapolis is 23 -- this equals the cost along the path connecting these two cities, so an arrow is placed that points to Indianapolis. The optimal path continues through Salt Lake City and Denver to Los Angeles.

It can be seen how this problem resembles the method of dynamic programming. If the problem is to be put into matrix form, one row of the matrix would be solved for. It would be necessary to start at each of the cities and solve for the optimal costs to all of the other cities.

There is an equal number of steps in using algorithms A and D, (2n+2) -- where n is the number of intermediate cities. The storage requirements of algorithm D are greater because of the larger numbers used, but algorithm D, being more general, includes algorithm A as a special case. By taking a different approach than Dantzig, Moore has been able to dispose of much of the trial-and-error in Dantzig's algorithms.

F. LEE'S ROUTING ALGORITHM

C. Y. Lee (Ref. L-1) has developed an algorithm in which the cost is defined as the sum of the number of obstacles crossed and the number of intermediate cities. Lee states that his algorithm is a modification of Moore's algorithm A. Lee uses a cost of unity between all the points as Moore did. In his algorithm, Lee does add an additional

cost for crossing obstacles; however his algorithm works in the same manner as Moore's algorithm A. In going from one node to the next, the process is the same except that instead of simply counting the number of intermediate cities, the algorithm also keeps track of the number of times that an obstacle is crossed. These two costs are added to determine the final cost at each city.

In Fig. (2.11) three obstacles are shown on the map. Assume that it is again desired to travel from Los Angeles to Boston with the least cost. In the first step the numbers (0,0) are placed on Los Angeles. The first number in the expression is the distance travelled from Los Angeles, and the second number is the number of obstacle crossings on the path from Los Angeles to the current city. In the second step, all of the adjacent cities' costs are calculated: (1,0) is placed on San Francisco, (1,0) on Denver, (1,2) on Nashville, and (1,0) on Phoenix. The third step consists of putting (2,0) on Seattle, (2,1) on Salt Lake City, (2,2) on Oklahoma City, (2,2) on Savannah, (2,3) on Dallas, and (2,1) on New Orleans. The fourth step is to put a (3,0) on Fargo, (3,2) on Indianapolis, (3,1) on Dallas -- which replaces the (2,3) from the last step because 3+1 = 4 is less than 2+3 = 5, (3,1) at Miami, and (3,3) on Washington. The process continues until Boston is reached. The arrows are placed on the optimal path by retracing the steps taken to reach Boston.

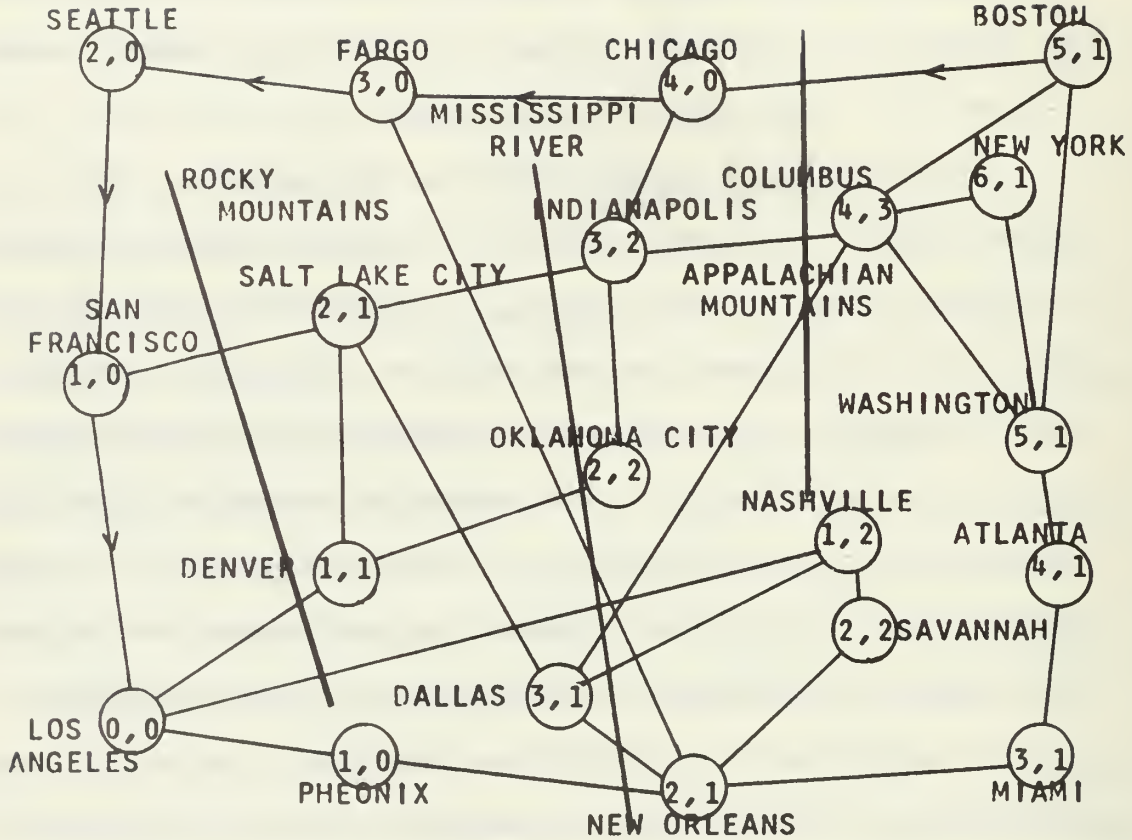


Fig. 2.11 The results of an application of Lee's algorithm.

Lee's algorithm has the same advantages as Moore's algorithm A and the added factor that it allows additional costs to be placed on a path. There is a price to pay for this added convenience, however; Lee's algorithm requires storage for two numbers as opposed to Moore's algorithm which has only one number for each of the cities.

G. SUMMARY

By comparing these algorithms it is found that each has its good and bad points. It seems that Dantzig's algorithms have little practical use because there are so many steps involved as well as large memory requirements.

The algorithms of Moore and Lee are very similar, but in comparison to Dantzig's algorithm they are more efficient. Because of this, the number of steps required is considerably less than in either Dantzig's algorithms or in direct enumeration, and the storage requirements are reduced.

There is great similarity between Moore's algorithm D and dynamic programming. The methods are different but the effect turns out to be the same. Dynamic programming is a more direct approach to the solution when a large number of points are involved.

III. LIM'S PATH-FINDING ALGORITHM AND ITS SIMULATION

A. INTRODUCTION

The algorithms already discussed assume that sufficient knowledge is available about the terrain and its features to obtain a cost matrix. Because of the lack of such knowledge in planetary exploration, a path-finding algorithm which uses only local sensor information has been developed by L. Y. Lim of the Jet Propulsion Laboratory (Ref. L-2). Given an initial point and a terminal point, the algorithm can find its way around obstacles whose size and extent are unknown. The path generated by the algorithm, however, may not always be the most efficient in terms of the total distance travelled, or time spent. By using one of the algorithms presented earlier, an optimal path can be obtained from the knowledge that is available from reconnaissance photographs. The robot follows this optimal path until it senses an obstacle from the local sensor information. With the local sensor information and Lim's path-finding algorithm, the robot finds its way around the obstacle and returns to the pre-determined optimal path.

Of the routing algorithms discussed earlier, dynamic programming seems to be the one best suited for this application. The speed and ease of calculation for a large

number of points makes it superior to the other algorithms. The dynamic programming algorithm has been programmed by Dr. D. E. Kirk in Fortran IV for operation on the IBM 7094 digital computer (see K-2). This program was converted for use on the IBM 360/67.

B. LIM'S PATH-FINDING ALGORITHM

Lim's path-finding algorithm assumes that:

1. all of the obstacles within the scanning range are detected;

2. initial and terminal points are given; and

3. the robot cannot pass over a predetermined elevation as a contour limit.

Lim's algorithm is designed to meet the following requirements:

1. if a path exists the algorithm can find it;

2. the algorithm navigates around all hazards; and

3. the algorithm uses a minimal number of memory locations.

Lim's path-finding algorithm consists of three parts, the main, left-scan, and right-scan algorithms. The main algorithm directs the robot straight ahead, or to the right or left. The right-scan takes the robot only to the right around an obstacle. The left-scan takes the robot only to the left around an obstacle. The right and left-scans enable the vehicle to navigate out of a box canyon or around other obstacles.

1. The Main Algorithm

a. Initially a vector is constructed from the starting point (X_o, Y_o) to the target point (X_t, Y_t). The azimuth of the vector, denoted by θ, is then determined.

b. The next point of travel (X,Y) is

$$X = X_o + r \cos \theta$$
$$Y = Y_o + r \sin \theta \qquad (3.1)$$

where r is the scanning range of the local sensors.

c. The point given in Eq. (3.1) is evaluated to see if it satisfies

$$F(X,Y) \leq \text{CONTOUR LIMIT} \qquad (3.2)$$

where F(X,Y) is the elevation at (X,Y). If the inequality (3.2) is satisfied, then the algorithm goes to step e; if not, the algorithm proceeds to step d.

d. If the point (X,Y) exceeds the contour limit, the algorithm scans to the left and right in one-degree increments (i.e., $\theta \pm 1$ degree, $\theta = \pm 2$ degrees, etc.) until an acceptable contour limit is found. If none is found, the vehicle is trapped and there is no way out; this condition can occur only at the starting point. The scanning process is: 1 degree to the left of θ, 1 degree to the right of θ, 2 degrees to the left of θ, 2 degrees to the right of θ, etc.

e. If the elevation F(X,Y) is within the contour limit at step c, it is necessary to determine if the terminal point has been reached. If the terminal point has been reached, then the job is finished; if not, (X_b,Y_b), which is the point before (X_o,Y_o), is replaced with (X_o,Y_o) and (X_o,Y_o), is replaced with (X,Y). If

$$X = X + r \cos \theta$$
$$Y = Y + r \sin \theta \qquad (3.3)$$

is acceptable, then the right and left counters are set to zero. If the point (X,Y) is found to be acceptable by deflecting to the right, then the right counter is set to one. If an acceptable point is found by deflecting to the left, then the left counter is set to one. If the point (X,Y) is found by deflecting to the left or right, then the distance to the target is calculated and stored as D_t. If both counters are not equal to one, then the algorithm returns to step a. If both counters are equal to one, the distance from the right point is compared with the distance from the left point. The distance which is the smaller determines whether the main algorithm goes into the left or right-scan algorithm.

2. The Left-Scan Algorithm

Because of the similarity between the left and right-scan algorithms only the left scan will be discussed.

a. Upon entering the left-scan, a target vector is constructed by using (X_o,Y_o) and (X_t,Y_t). A crawling vector is also constructed using (X_b,Y_b) and (X_o,Y_o). The azimuth of the target vector and the crawling vector are then found. The crawling vector azimuth is called $\theta 1$. The distance from (X,Y) to (X_t,Y_t) at the time of detecting the obstacle is D_{MIN}.

b. The next point of travel (X,Y) is

$$X = X_o + r \cos(\theta 1 - 90^o)$$
$$Y = Y_o + r \sin(\theta 1 - 90^o) \qquad (3.4)$$

where r and $\theta 1$ are defined as before. An evaluation of the elevation F(X,Y) is then made. If F(X,Y) is not within the contour limit, then $\theta 1$ is incremented by one degree intervals until an acceptable contour limit is found. Once an acceptable contour limit is found, the distance to the target is calculated by using (X,Y) and (X_t,Y_t). This distance is set equal to D_t. If D_t is not less than D_{MIN} then the algorithm returns to step a of the left scan algorithm. If D_t is less than D_{MIN}, then the contour value F(X,Y), where

$$X = X_o + r \cos \theta$$
$$Y = Y_o + r \sin \theta \qquad (3.5)$$

is evaluated.

If F(X,Y) is greater than the contour limit, the algorithm returns to step (1) of the left-scan algorithm. If F(X,Y) is less than the contour limit, the algorithm resets the left and right counters to zero and the algorithm returns to the main section.

C. DEVELOPMENT OF TERRAIN SIMULATION AND CONTOUR MAPPING

With the introduction of a path-finding algorithm it is necessary to test the algorithm on simulated terrain. Since the simulation is to be topographical, it must have various levels. The simulation was performed by using Gaussian density functions to represent the hills and mountain ranges that would be found by photoreconnaissance of the actual planet. The elevation at a point of the terrain is found simply by summing the Gaussian density functions.

Two different types of equations using Gaussian density functions were developed in order to simulate the different shapes of mountains. Mountains were assumed to be either circular or elliptical. The equation used to simulate mountains of the circular type (Fig. 3.1) was:

$$\text{CONTOUR} = \text{HEIGHT} * \text{EXP}(-1.((X - X_o)/\sigma_X)^2) + ((Y - Y_o)/\sigma_Y)^2) \; ; \tag{3.6}$$

where CONTOUR is the height of the mountain at (X,Y). (X_o,Y_o) is the coordinate of the center of the circular mountain;

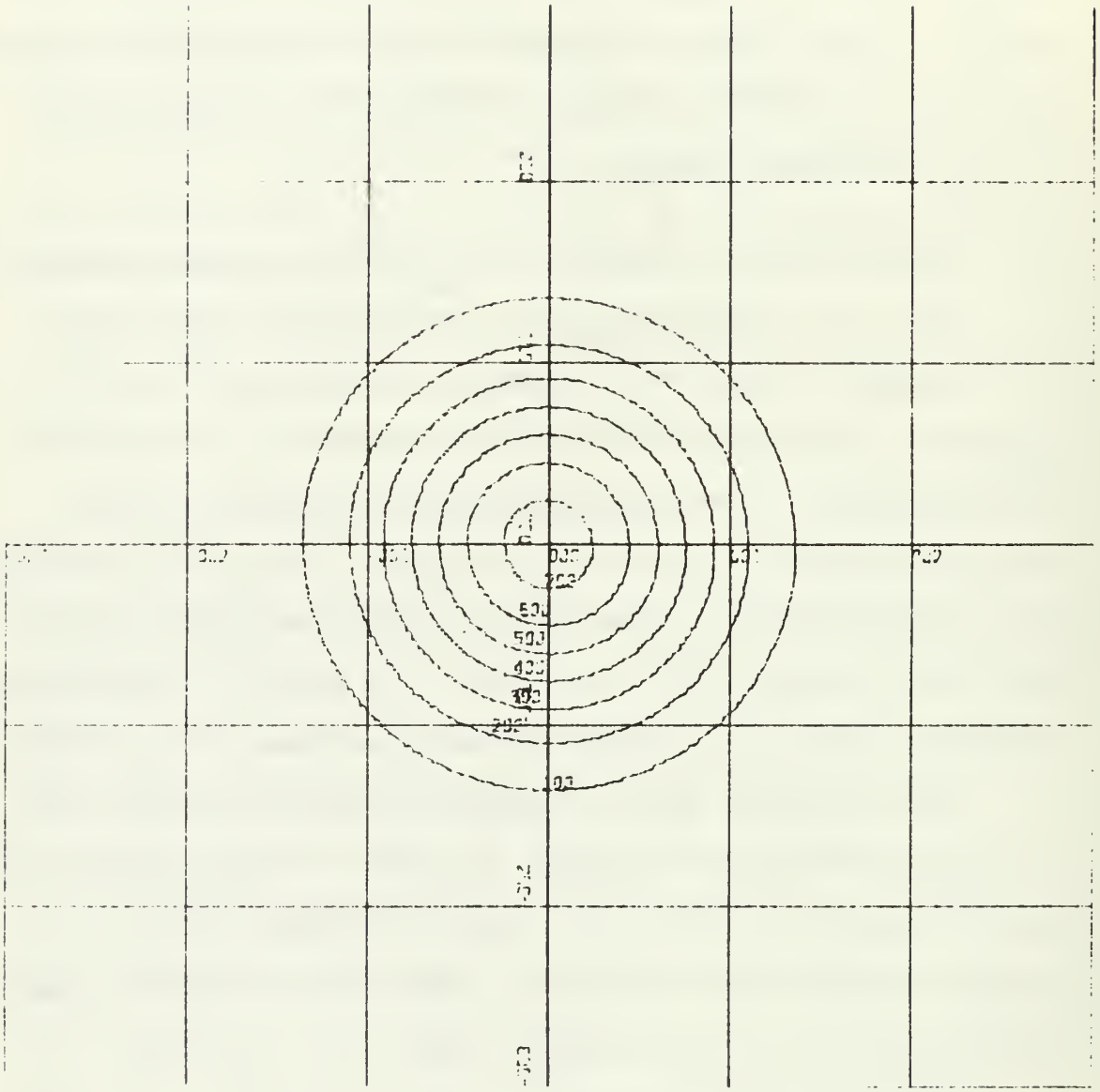


Fig. 3.1 Contour map of a circular hill.

Height is the height at the center point (X_o, Y_o);

σ_X is the standard deviation in the X direction; and

σ_Y is the standard deviation in the Y direction, and equals σ_X for the circular case.

The equation used to simulate the mountain with an elliptical contour as shown in Fig. (3.2) is:

$$\text{CONTOUR} = \text{HEIGHT} * \text{EXP}(-1.((((X - X_o)\cos(\text{ALPHA}) + (Y-Y_o)\sin(\text{ALPHA}))/\sigma_X)^2 + ((-1.(X-X_o)\sin(\text{ALPHA}) + (Y-Y_o)\cos(\text{ALPHA}))/\sigma_Y)^2)). \quad (3.7)$$

Where ALPHA is the angle from the X axis to the semi-major axis of the ellipse. By varying the sigmas, the height, and the angle ALPHA, it is possible to simulate a variety of terrain features.

Table (III.1) gives some typical mountain heights observed on lunar charts. The mountains are generally circular in nature. The craters are due in part to volcanic action as well as meteors crashing into its surface. These craters were simulated by adding two Gaussian functions together. One hill -- the larger of the two -- was added to a hill with a negative height. The radii were, as indicated in Table (III.1), simulated by setting the sigmas equal to the radius divided by 2.5.

In order to roughen the terrain, an additional feature was included to create obstacles on the terrain map. Two different formulations were contrived, one for the circular

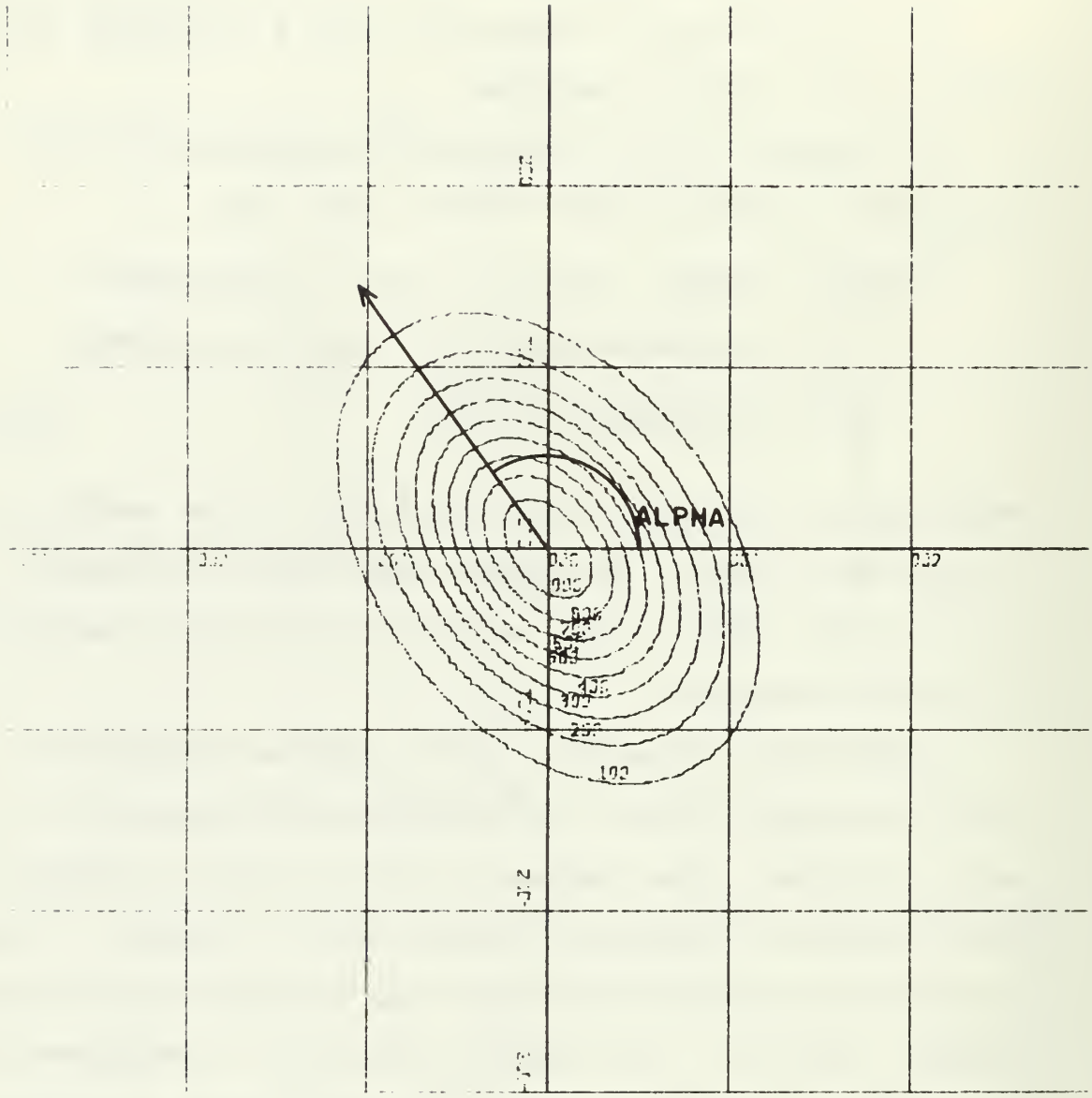


Fig. 3.2 Contour map of an elliptical hill.

mountain and another for the elliptical mountain. The number of obstacles was assumed to be three times the height of the mountain. It was natural that the obstacles would be placed at irregular intervals and that they would vary in size.

In the case of the circular mountain, an imaginary set of axes was drawn through (X_o,Y_o) parallel to the north and east axes of the grid. The position angle of the obstacle was determined by a uniform-distribution random number generator. Once the angle had been determined, the position on this vector is based on a Gaussian distribution with its mean equal to the radius of the hill. It was assumed that there is debris from the top of the mountain to a distance of two times the radius of the mountain. This means that the sigma of the Gaussian distribution is equal to the sigma of the mountain. In keeping with the rest of the terrain simulation, the obstacle was assumed to be a circular Gaussian hill with a radius equal to the height of the obstacle. This meant that the sigma of the obstacle is equal to the obstacle height divided by 2.5. The height of the obstacle was determined by assuming that the larger the obstacle, the farther it would roll. The maximum obstacle height is assumed to be one one-hundredth of the height of the mountain. As stated earlier, the obstacles are assumed to be confined to within a radial distance of two times the radius of the mountain. Under these conditions the equation

Name	Height in Meters	Radius in Meters
Encke γ	930	3100
Encke σ	720	2200
Maestlin μ	810	3200
Kepler	700	3200
Flamsteed	1670	42000
Encke B	1100	600
Bessarion	900	1000
Marius A	2000	40000
Tobias Mayer C	420	2800
Mickhius β	750	4000
Unknown	1020	4100
Unknown	690	3800
Unknown	1010	5000
Marius D	100	900

Table III.1

Dimensions of Some Lunar Mountains

$$H_1 = \frac{H_o}{100*2R} * D_1 \qquad (3.8)$$

was obtained.

H_1 is the height of the obstacle;

H_o is the height of the mountain;

R is the radius of the mountain; and

D_1 is the distance of the obstacle from the center of the mountain as determined by the Gaussian distribution. The results of this theory are shown in Fig. (3.3).

The formulation for the placement of debris about the elliptical mountains was done in a similar manner as that for circular hills. Because the hill is not symmetric, large amounts of mass were assumed to lie along the semi-major axis. Since the majority of the mass has moved along the semi-major axis, it is reasonable to assume that the debris created by this mass movement will be in this direction.

The total number of obstacles was again selected as being equal to three times the height of the mountain. A set of imaginary axes was drawn to coincide with the semi-major and semi-minor axes of the ellipse. It was assumed that σ_X was greater than σ_Y; in some cases this requires the X and Y axes to be completely independent of the north and east axes of the grid. The angle BETA, as indicated in Fig. (3.4) was determined by the arctan of σ_Y/σ_X. The other lines defining the four segments were found by

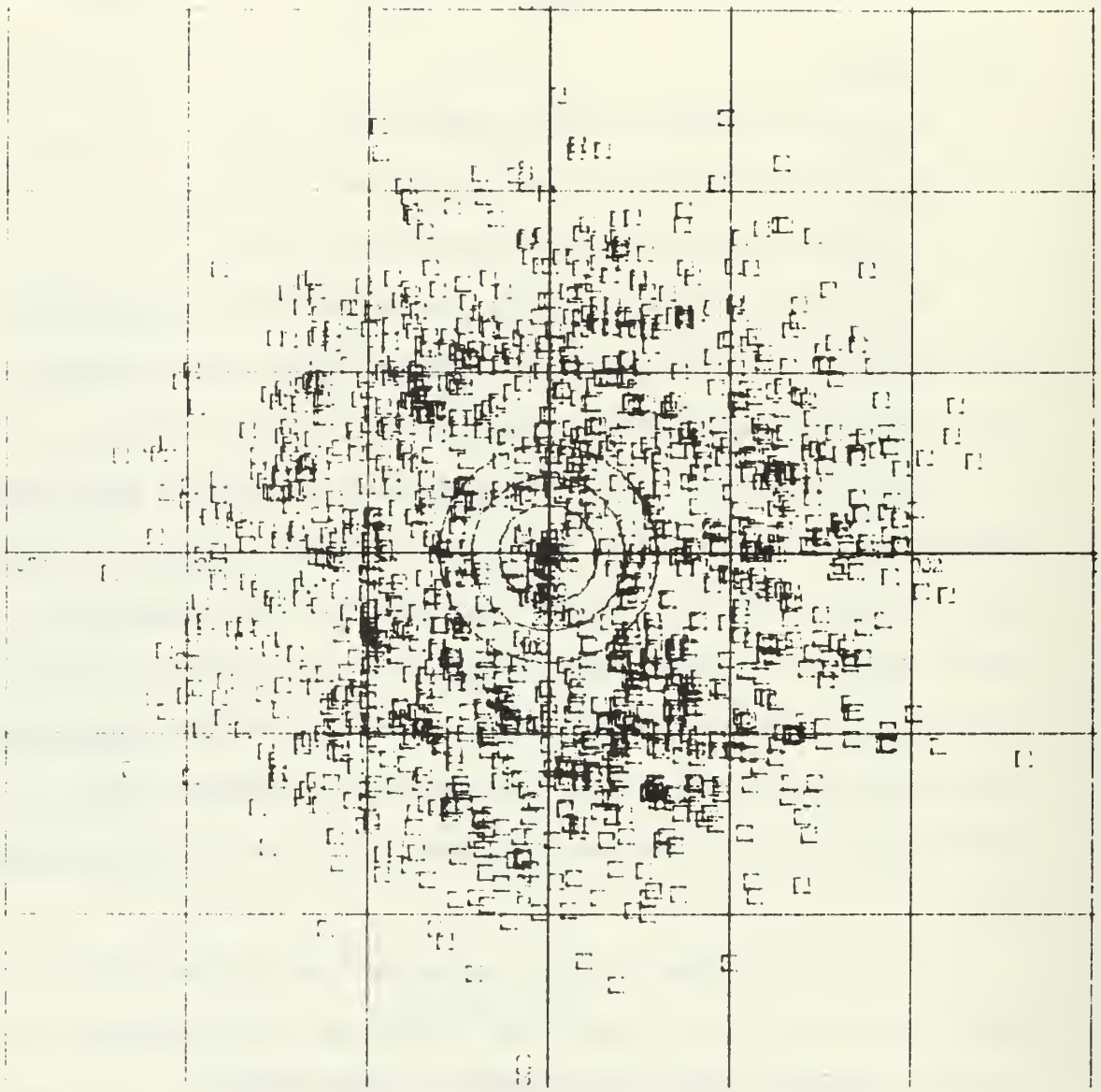


Fig. 3.3 Distribution of obstacles around a circular hill.

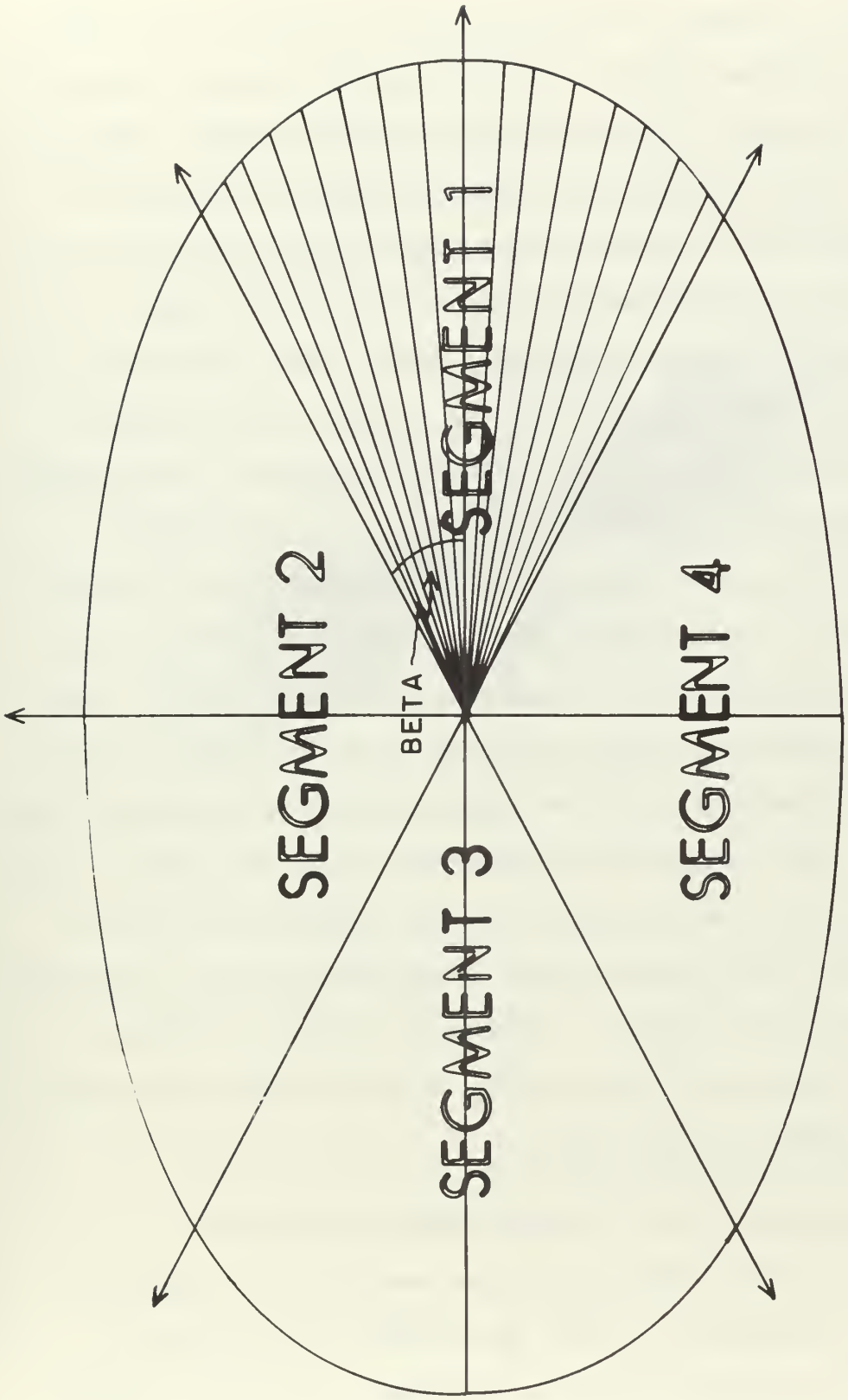


Fig. 3.4 The sections of an elliptical hill used in determining the disposition of debris.

adding 180 degrees and subtracting 360 degrees from BETA. Once the ellipse had been divided in this manner, the number of obstacles in each segment was determined. Segments one and three contained the largest number of obstacles. Each segment was divided into sixteen equal sections with the number of obstacles in each section being equal for a given segment. Sixteen separate radii were calculated by taking the difference between the longest and shortest radii of the ellipse and dividing by sixteen. The sixteen radii were used as the means of Gaussian distributions which determine the obstacle's coordinates along the position vector. The sigma of the Gaussian distribution was equal to the radius of the mountain divided by 2.5. The position vector was generated from a random number generator which provided a uniform distribution within each of the sections. The Gaussian distribution along the position vector determined the distance from the center of the hill. The height of the obstacle was determined by Eq. (3.8) and the sigma of the obstacle equaled the height divided by 2.5. The results of this theory of positioning the obstacles are shown in Fig. (3.5).

D. SIMULATION OF LIM'S PATH-FINDING ALGORITHM

Lim's path-finding algorithm has been programmed in Fortran IV language for the IBM 7090/94 IBJOB System. Before the algorithm could be applied, a suitable terrain simulation had to be generated. Five Gaussian hills were

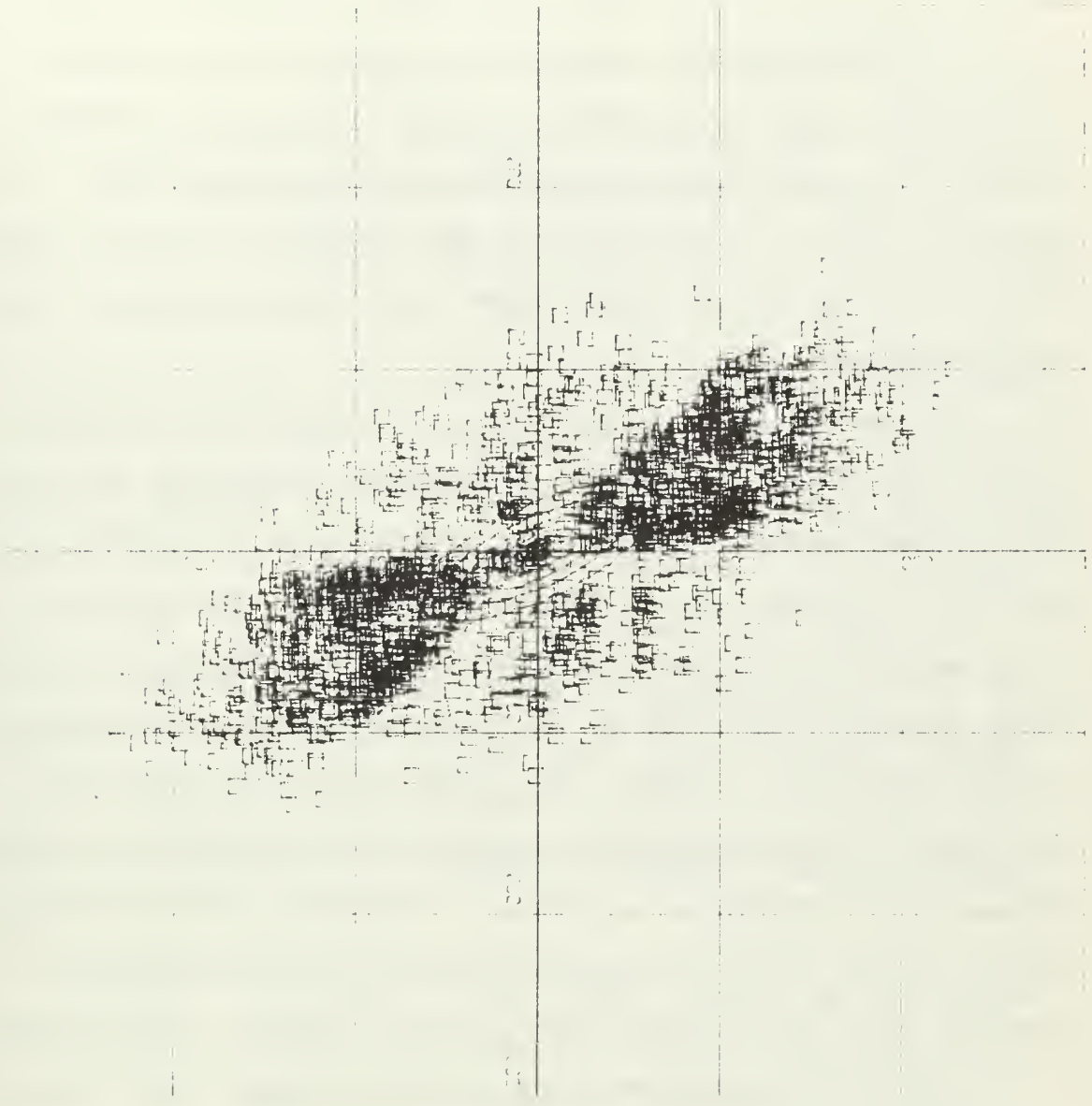


Fig. 3.5 Distribution of obstacles around an elliptical hill.

used and the obstacles were generated around each of them by using the procedure described in the previous section. The contour plotting package, written by Dr. M.O. Dayhoff (Ref. D-2), shows only the elevations generated by the Gaussian hills. The heights of the obstacles were not included because of the large number of calculations and the long computer times.

The next step in preparation for running Lim's algorithm was to use the dynamic programming algorithm to find the optimal paths. The map was divided into 81 grid points that were equally spaced 2500 meters apart. The cost matrix used in the dynamic programming algorithm was the actual distance travelled by the vehicle over a three-dimensional Euclidean surface. The node matrix was calculated and used in determining the optimal path for Lim's algorithm. In the first attempts to use Lim's algorithm the elevation limit proved to be too restrictive and the route wandered too far from the optimal path given by dynamic programming.

The first corrective action was to change the elevation limitation to a slope limitation. A slope of 11 degrees proved to be satisfactory for later simulations. Although the slope limitation allowed greater maneuverability for the algorithm, one big disadvantage became apparent. It was more difficult to see why the algorithm reacted as it did to certain situations. With the slope limitation, it was impossible to draw a wall around an

obstacle as could be done with the elevation limit. To alleviate this difficulty, a factor was added to the section of the dynamic programming algorithm that calculates the cost matrix from the simulated terrain. Every time a path crossed a slope limitation, the cost for that path was multiplied by this factor. This factor was tried at two different values. The first of these was 2.0, which means that the robot would go twice as far to avoid a slope limitation. Because the factor was small, the paths calculated by the dynamic programming algorithm went over the slope limitation. In simulations one and two, the number of points used in the dynamic programming algorithm were reduced to sixteen with intervals of 6666.67 meters between nodes. This reduction was due to the fact that with 81 points on the grid the length of computer time was excessive.

In Fig. (3.6), the nominal path calculated by dynamic programming crosses a hill which surpasses the slope limitation. The algorithm was unable to follow the nominal path and had to calculate its own path around the hill. The contour of the hill forced the algorithm to make such a large deviation from the nominal path.

In Insert B of Fig. (3.6) the algorithm performed a switchback movement. In Lim's original algorithm this was not possible because the test to get out of the right or

left scan mode not only required that the distance to the target be less than D_{MIN} but also that the point (X,Y) had to be

$$X = X_o + r \cos \theta$$
$$Y = Y_o + r \sin \theta \ . \qquad (3.9)$$

Frequently, a point was found such that the distance to the target was less than D_{MIN}, but the point (X,Y) was beyond the slope limitation and failed to meet the second requirement to get out of the scan algorithm. As a result, the algorithm would continue around the same obstacle until it came back to the same point where the distance to the target was less than D_{MIN} . At this point the same calculations would re-occur and the algorithm would continue around indefinitely. By eliminating the requirement that (X,Y) be generated as given in Eq. (3.9) the algorithm was able to move to the next obstacle and proceed around it in the usual fashion.

Fig. (3.7) and Inserts A and B of Fig. (3.7) show the results of simulation run one with the obstacles; the robot went around them in excellent fashion. By comparing Fig. (3.7) and Inserts A and B of Fig. (3.7) with Fig. (3.6) and Inserts A and B of Fig. (3.6) it is seen that there is little variation in the two paths. This shows that the amount of deviation from the optimal path is dependent on the size and slope of the obstacle on the path.

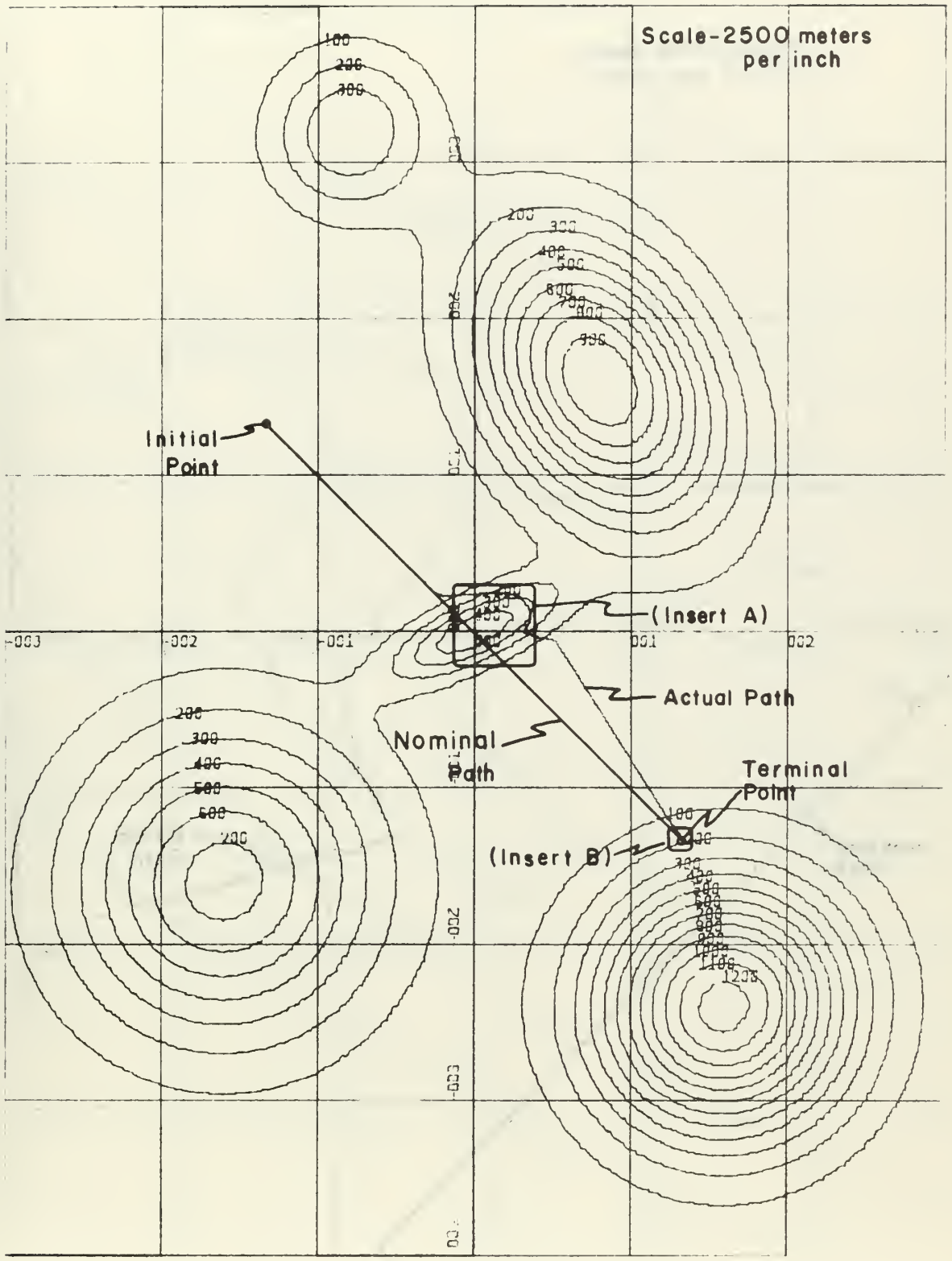
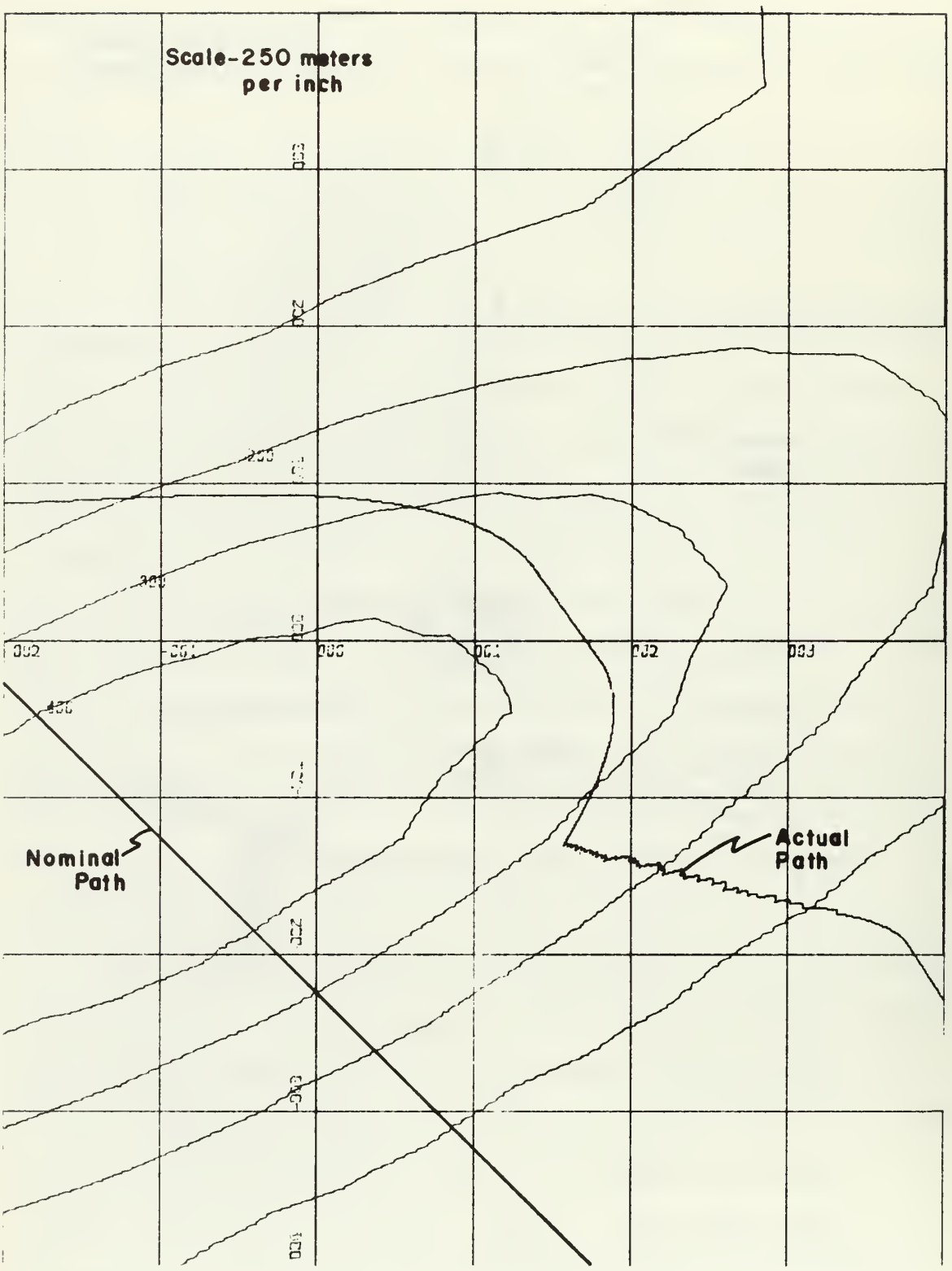
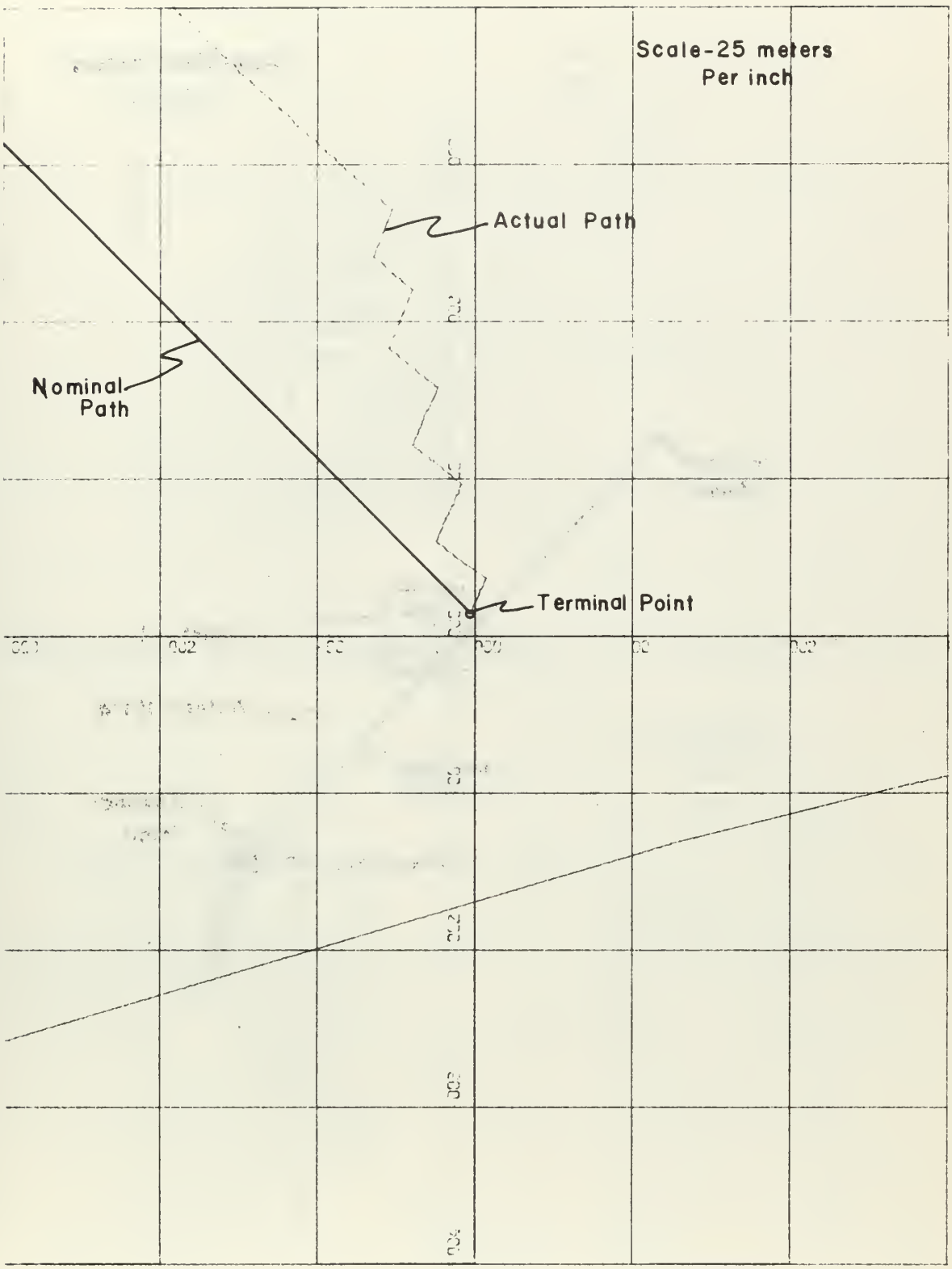


Fig. 3.6 Simulation one without the obstacles.



Insert A of Fig. 3.6 Simulation one without the obstacles.



Insert B of Fig. 3.6 Simulation one without the obstacles.

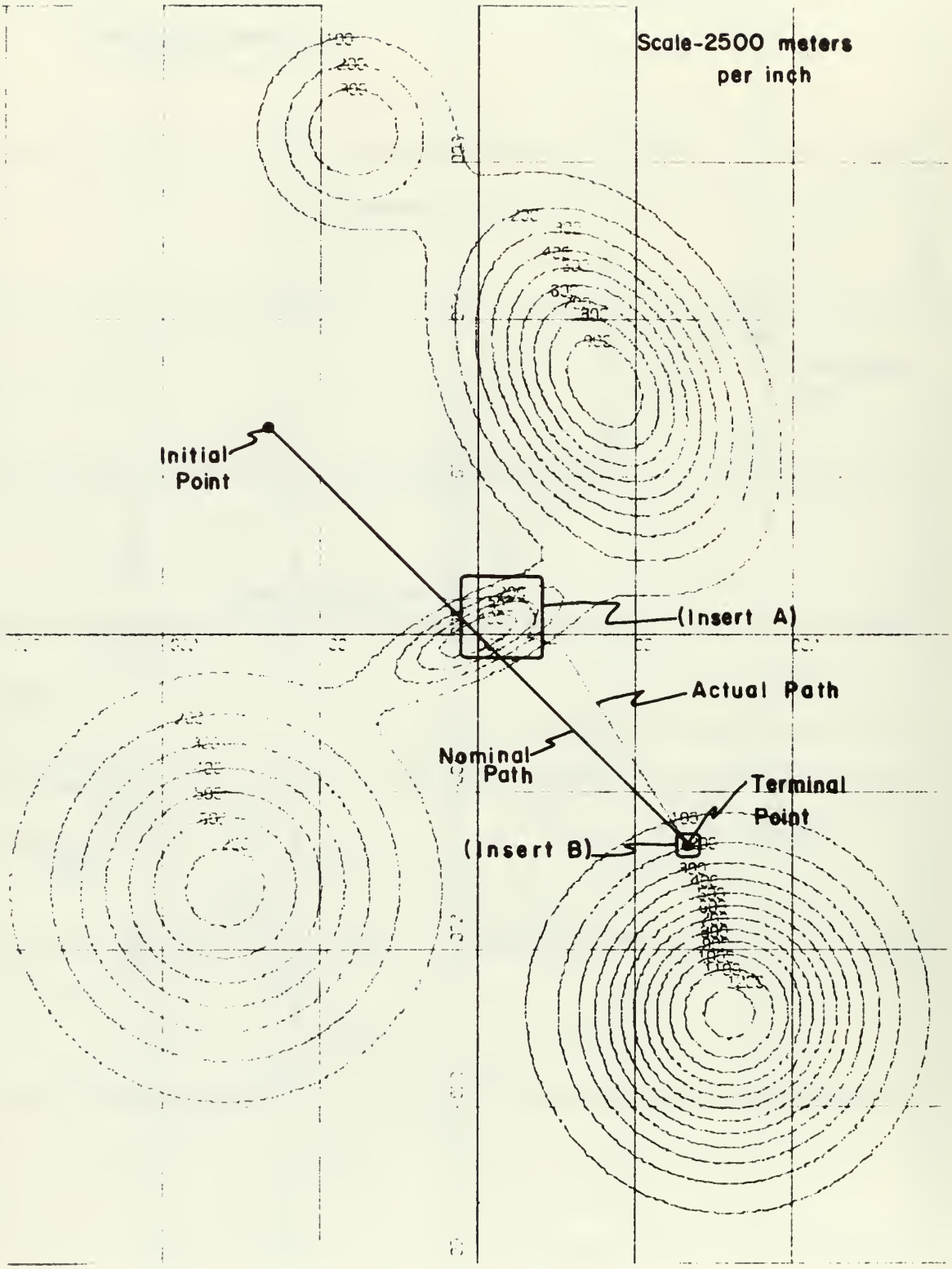
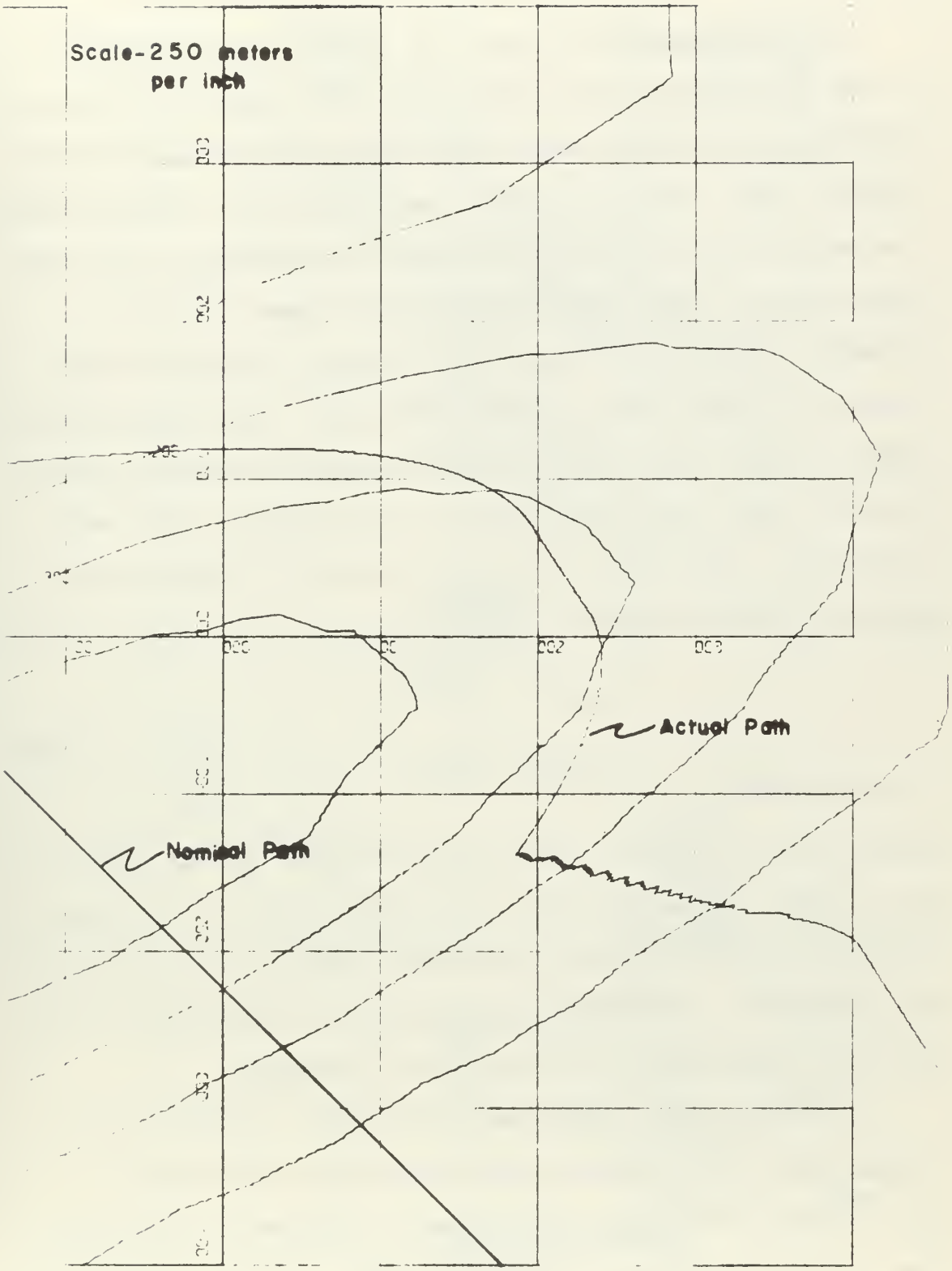


Fig. 3.7 Simulation one with the obstacles.



Insert A of Fig. 3.7 Simulation one with the obstacles.

In simulation two (Fig. 3.8) there is no difference in the simulations with and without the obstacles which indicates that although there are obstacles, none lies on the optimal path. This run, like simulation one, deviated from the optimal path because the dynamic programming algorithm again chose a path which crossed the slope limitation. Another interesting feature did come up, and can be seen in Insert A of Fig. (3.8) in the small box. The algorithm had taken a route that went in circles until it found a satisfactory point that would allow it to leave the left-scan algorithm. The difficulty is due to the 90 degree factor present in the equation

$$X = X_o + r \sin(\theta 1 \pm 90^o)$$
$$Y = Y_o + r \cos(\theta 1 \pm 90^o) \qquad (3.10)$$

when in the scan mode. With the 90 degree factor added in Eq. (3.10), the algorithm would often continue in circles indefinitely. With the 90 degrees reduced to 45 degrees, the algorithm never continued in circles indefinitely and the algorithm went in circles less frequently.

In simulations three and four (Figs. 3.9 and 3.10) a small square 400 meters by 400 meters was taken out of the center of the map. The dynamic programming algorithm was run for 81 points, 50 meters apart. A cost factor of

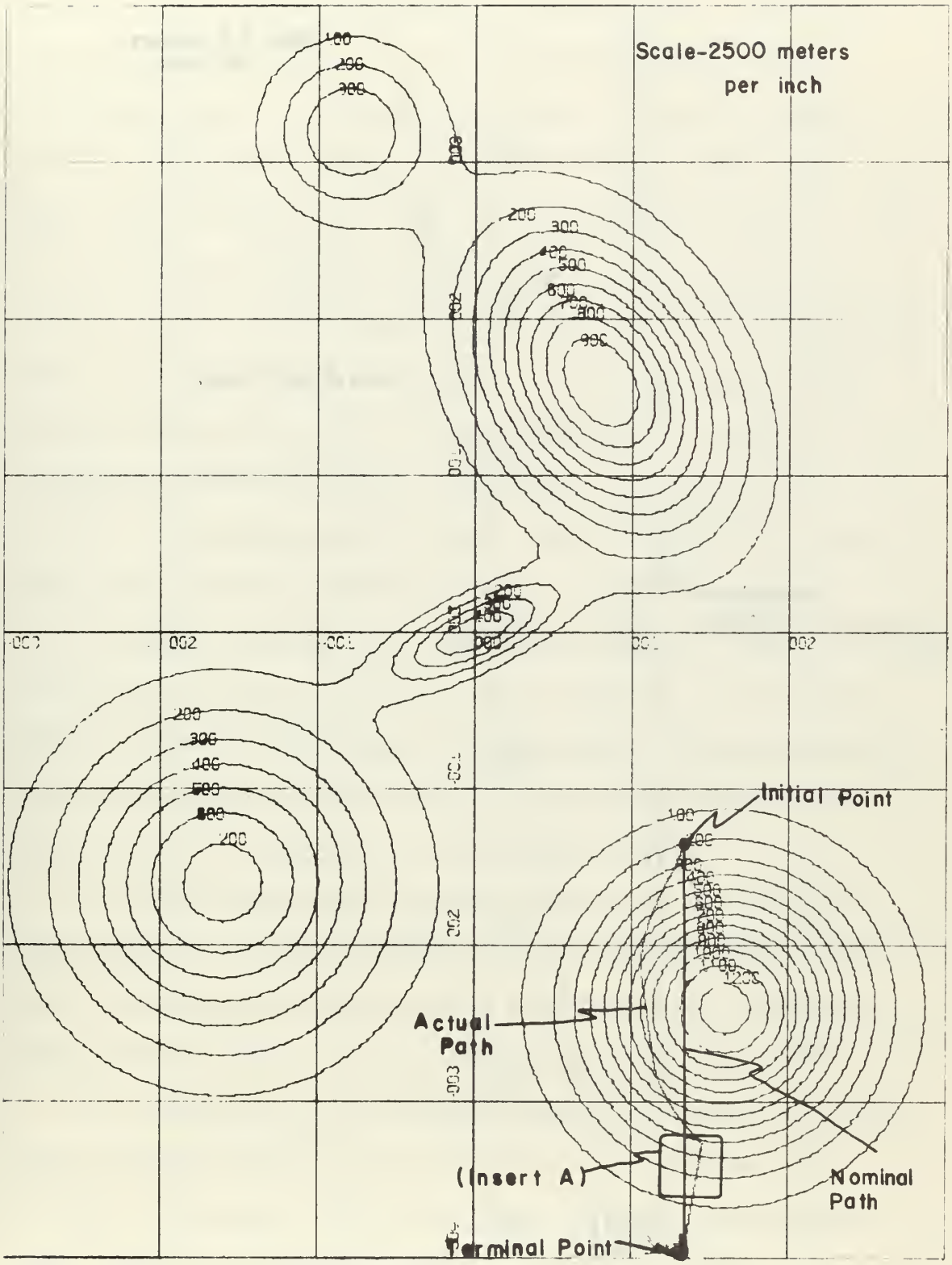
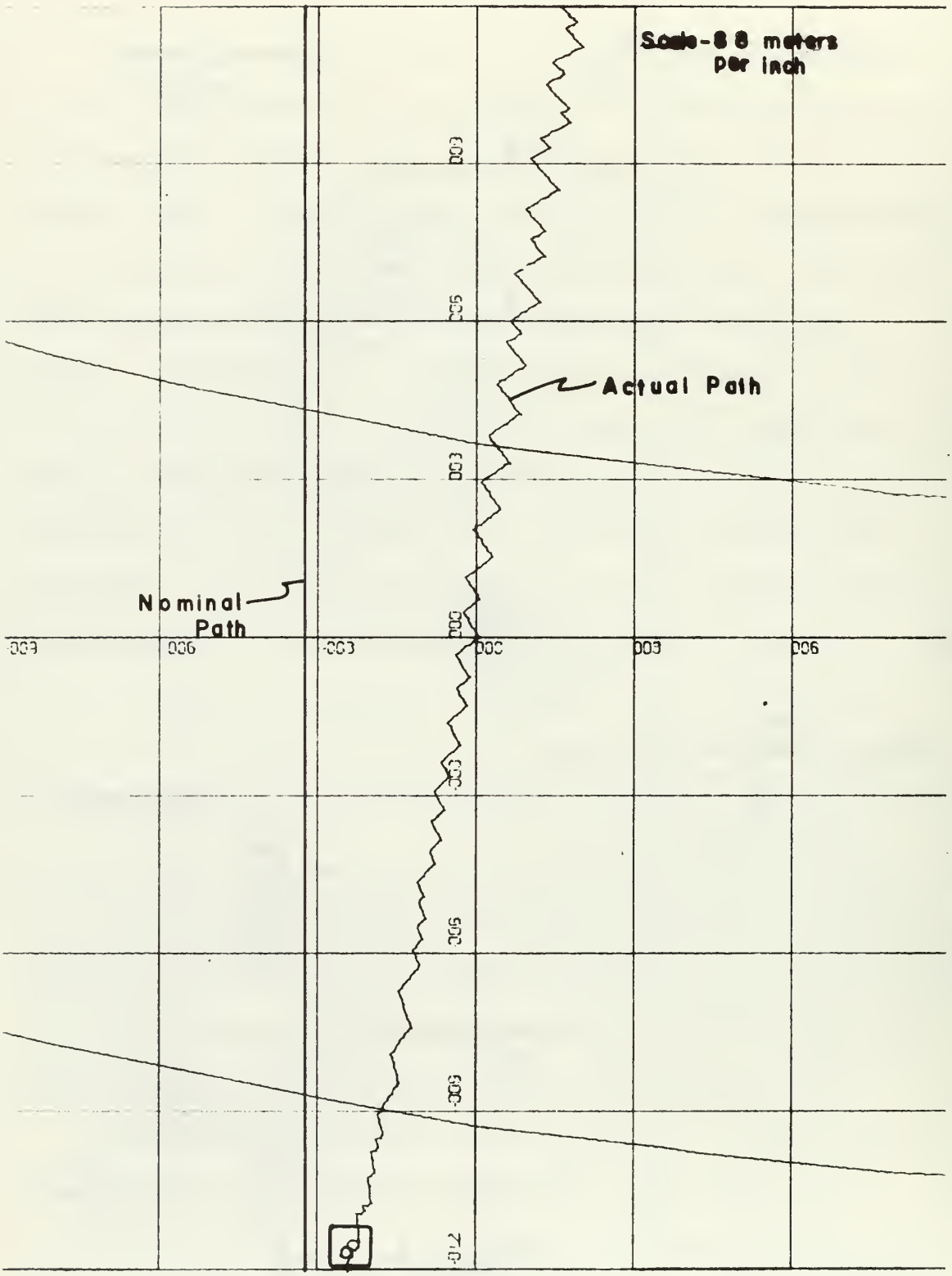


Fig. 3.8 Simulation two.



Insert A of Fig. 3.8 Simulation two.

2.0 was used in determining the cost matrix. The paths selected by the dynamic programming algorithm went over the slope limitation because the cost factor was small.

In simulation three (Insert A of Fig. 3.9) when the algorithm was about to reach the intermediate node B given by the dynamic programming solution, the algorithm hit a slope limitation and took a long and devious path to get back to node B. When the robot is so close to the node point a provision should be incorporated to allow it to go to the next node, because the purpose of the algorithm is to reach the final destination. The intermediate nodes are only to direct the algorithm along a nominal path.

As the algorithm was progressing towards node F it encountered a slope limitation that could not be crossed without going out to point Z (Fig. 3.9). This occurrence can only be explained by the fact that the slope of the hill caused the algorithm to take this path.

In simulation four the path chosen by the dynamic programming algorithm crossed the slope limitation for a short distance from the initial point; hence the algorithm stayed close to the nominal path.

In simulations five through eight, the cost of going over a slope limitation was increased to 1000 when the dynamic programming algorithm was executed. With this high cost any path that crossed a slope limitation was not chosen as a nominal path. In simulations seven and

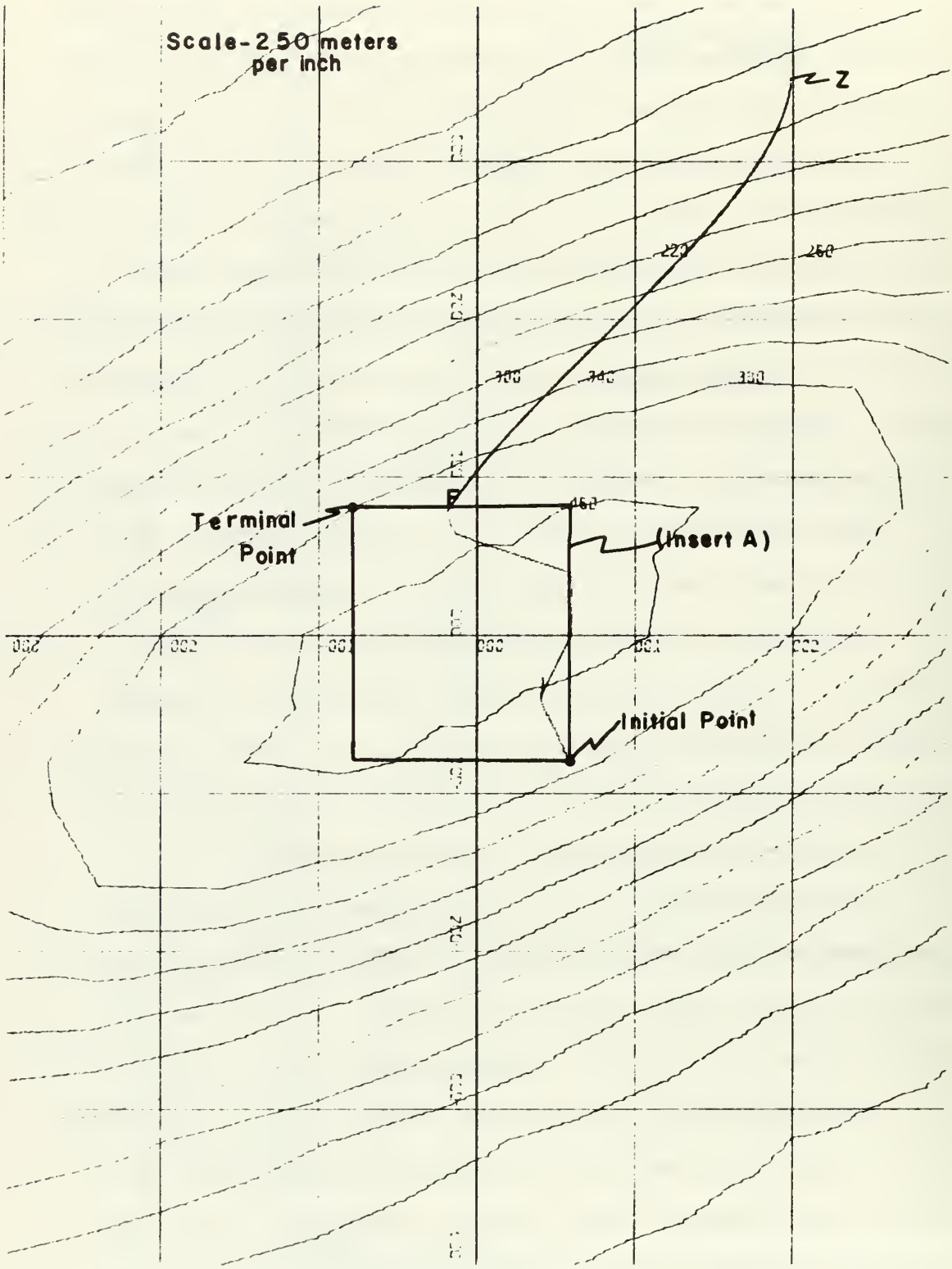
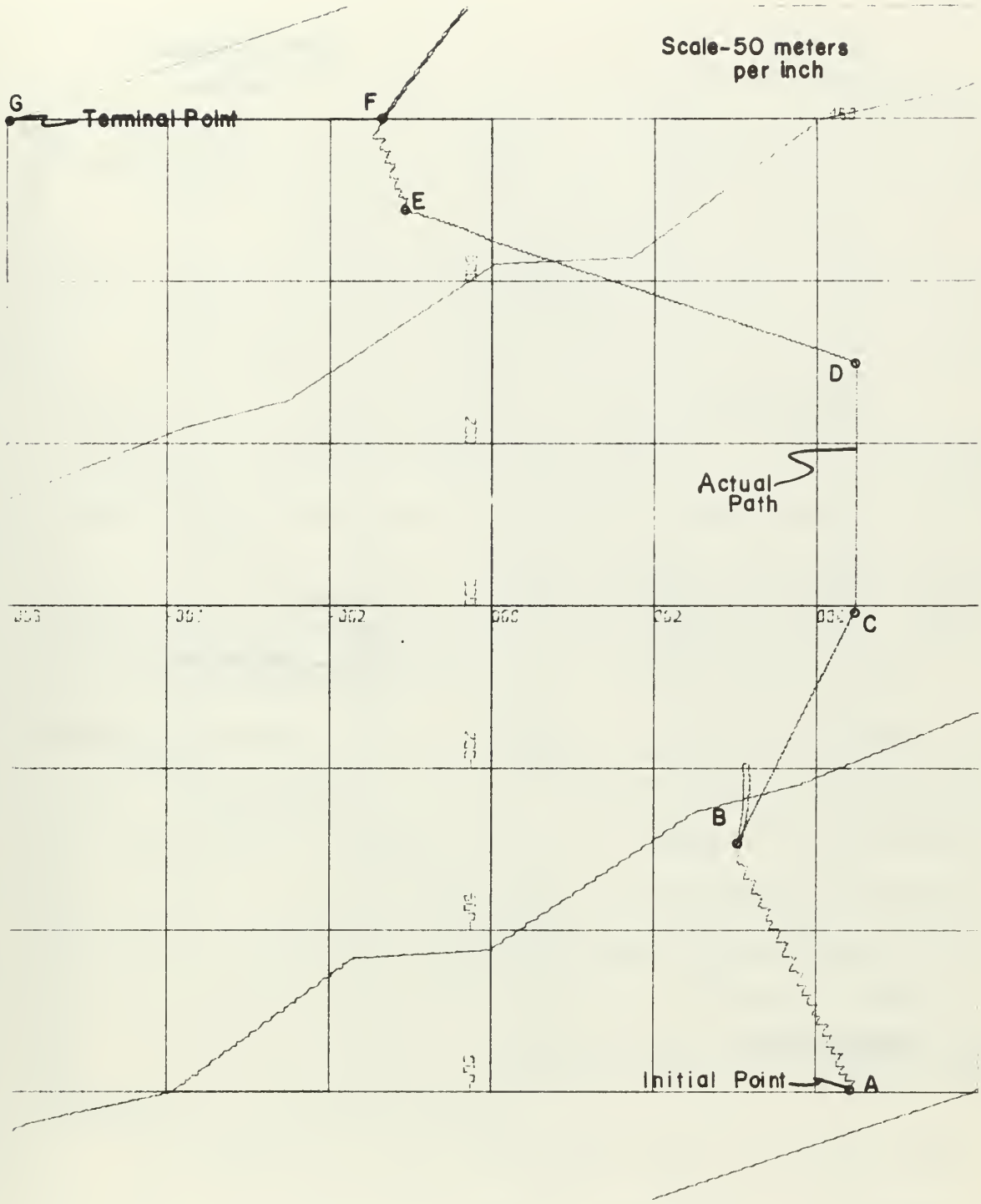


Fig. 3.9 Simulation three.



Insert A of Fig. 3.9 Simulation three.

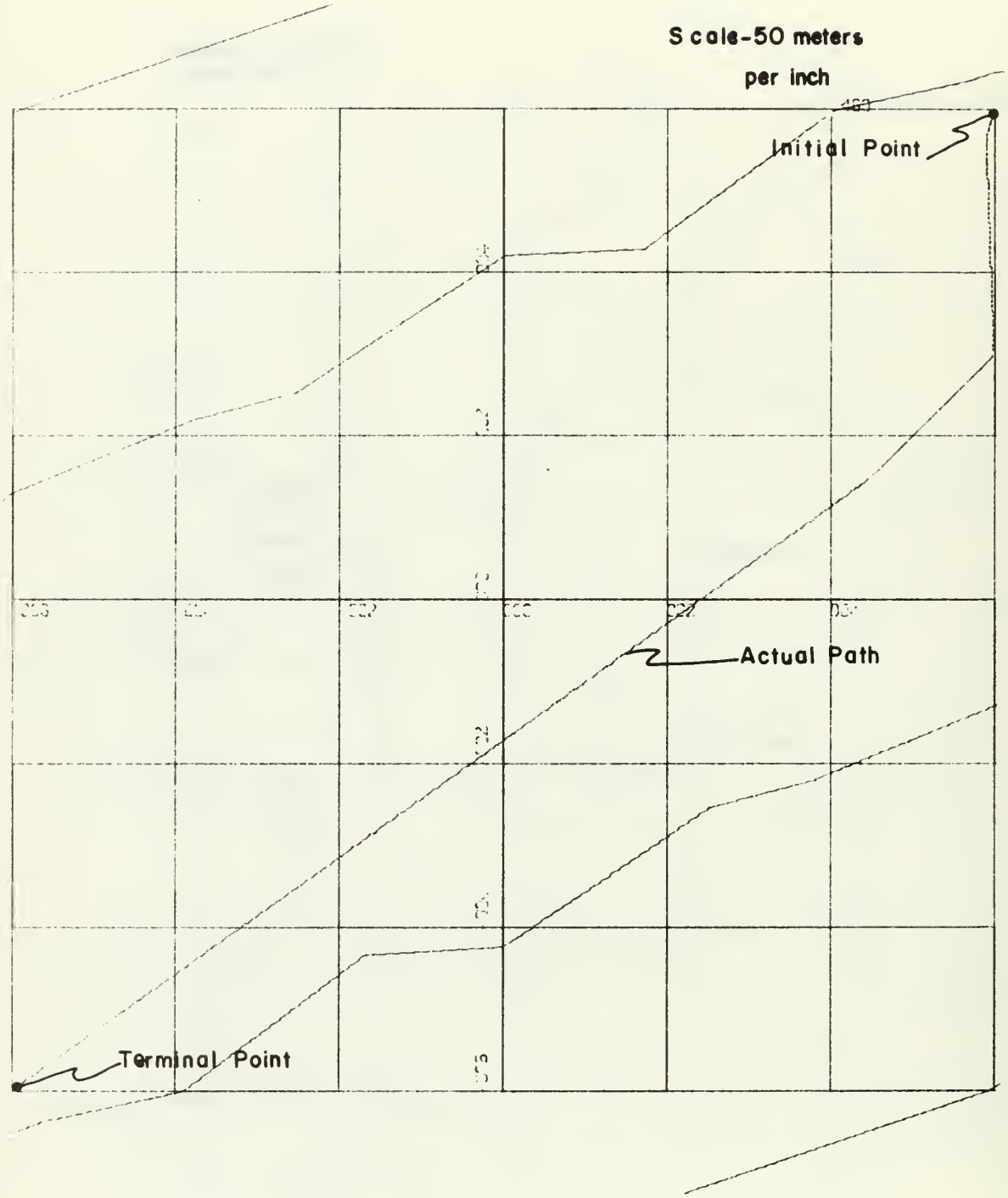


Fig. 3.10 Simulation four.

eight, 81 nodes were used (as in simulation three and four). All the simulations showed good results both in getting to the target point and in detecting and going around obstacles.

In Insert A of Fig.(3.11) a small obstacle was encountered, but, as can be seen, the path-finding algorithm made only a slight deviation from the nominal path. Insert B of Fig. (3.11) shows how the algorithm went to the right around an obstacle and returned itself to the optimal path.

Insert A of Fig. (3.13) shows that an obstacle was detected, and how the algorithm took the robot around the obstacle. Insert B of Fig. (3.13) shows how the robot went around an obstacle but did not return to the optimal path. Instead the algorithm took a straight-line path to the target. The same feature occurred in Insert C of Fig. (3.13). This feature of following a straight-line path to the target after the robot has gone around an obstacle may seem to be a good idea, but once the robot starts calculating its own path over a long distance the purpose of using dynamic programming has been defeated. Inserts D and E of Fig. (3.13) show how the robot went around the obstacle and returned to the optimal path.

In simulations seven and eight (Figs. 3.14 and 3.15) a devious route was calculated by the dynamic programming algorithm. Both simulations stayed on the nominal path even with the obstacles placed around the hill. This indicates that even though there were obstacles, none were on the nominal path.

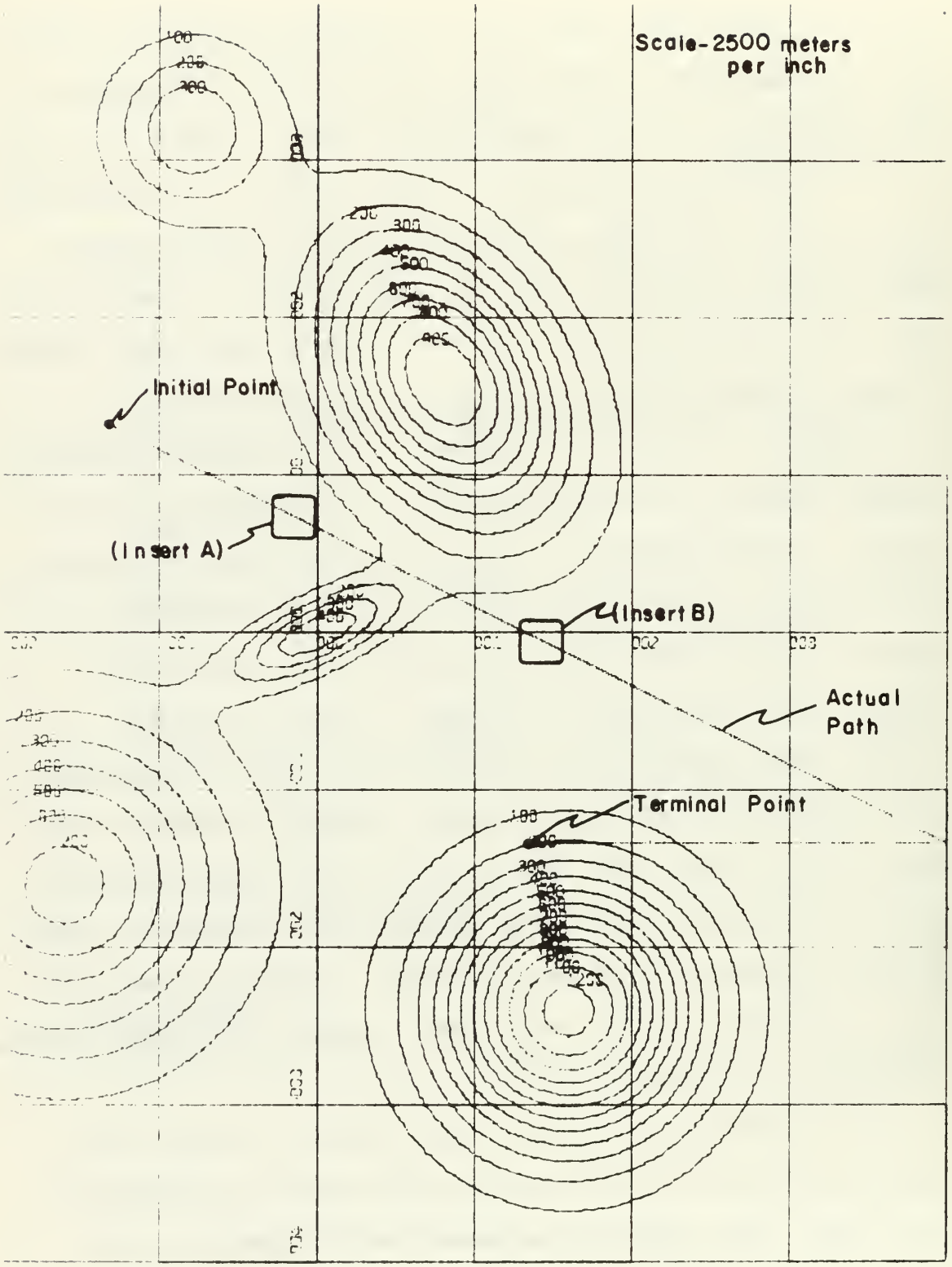
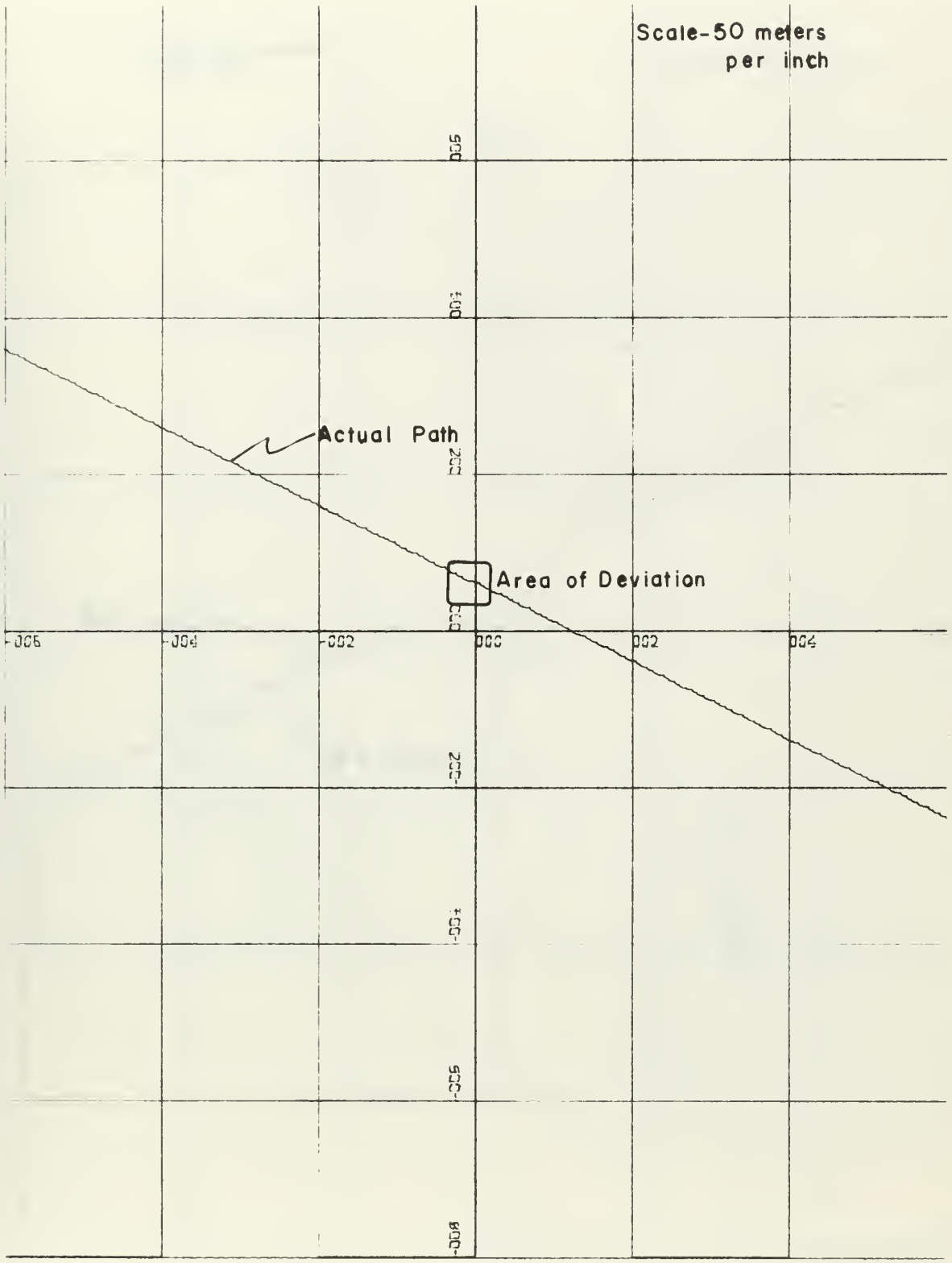
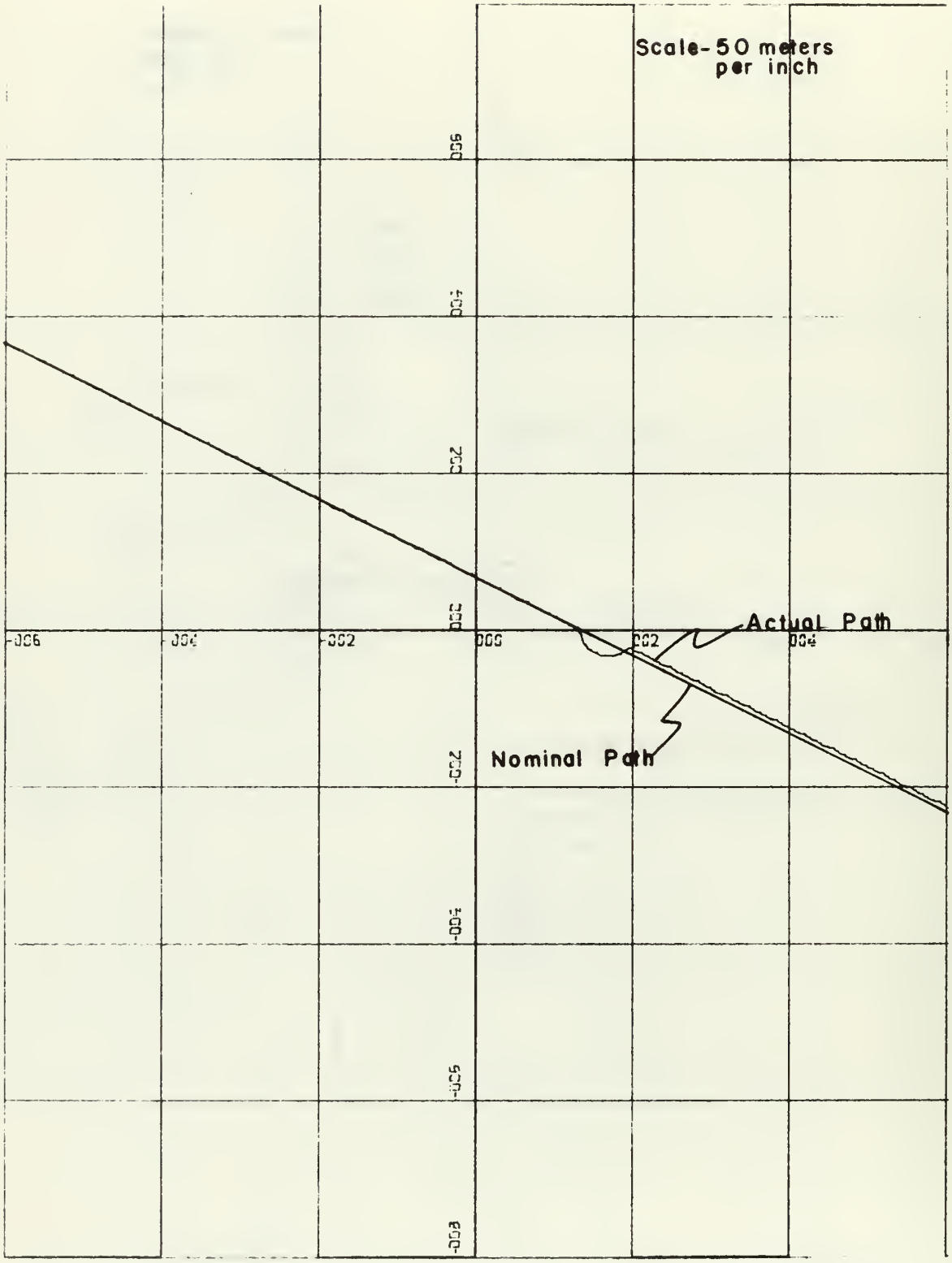


Fig. 3.11 Simulation five.



Insert A of Fig. 3.11 Illustrating the path taken around an obstacle.



Insert B of Fig. 3.11 Illustrating the path taken around an obstacle.

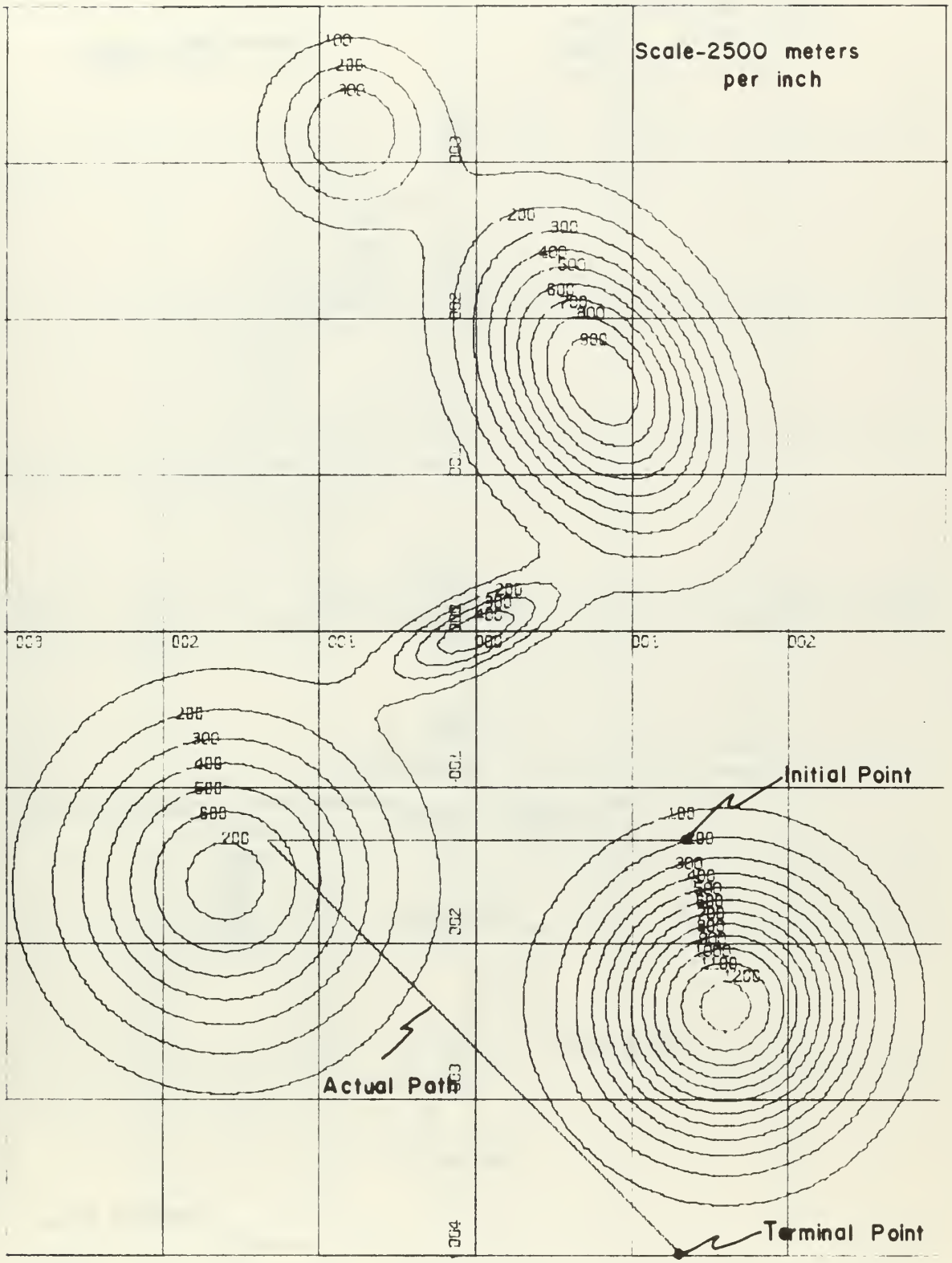


Fig. 3.12 Simulation six without the obstacles.

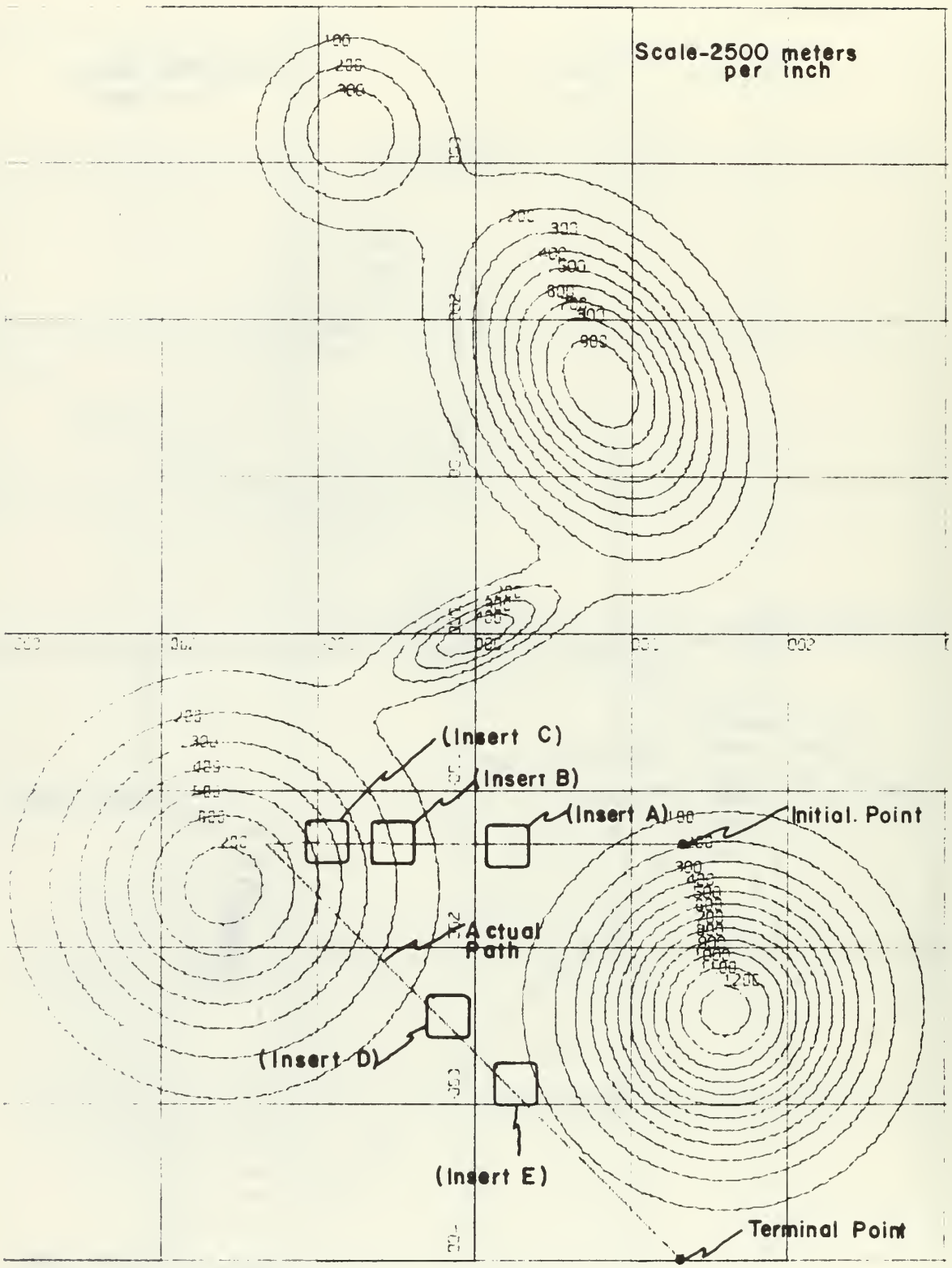
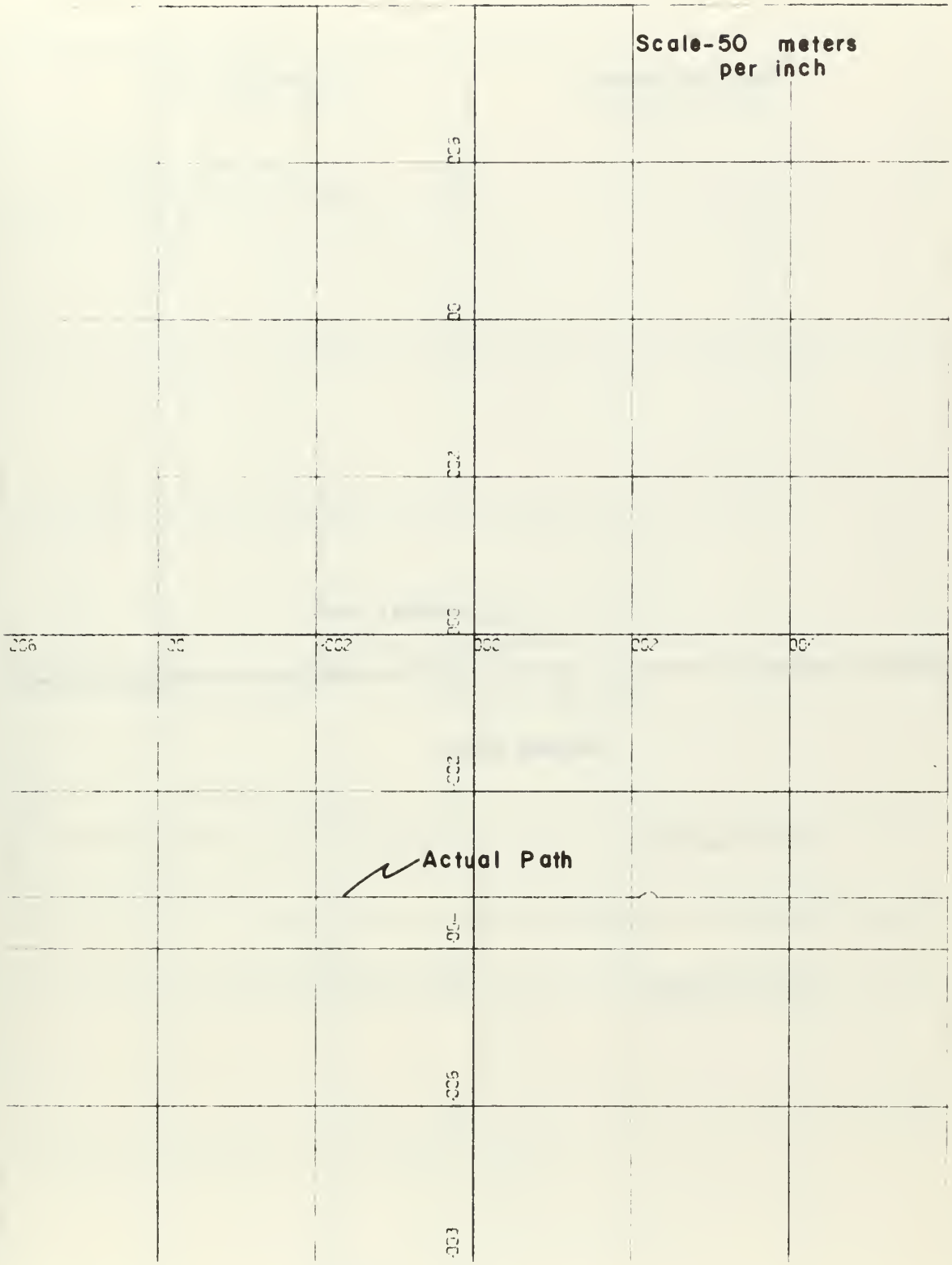
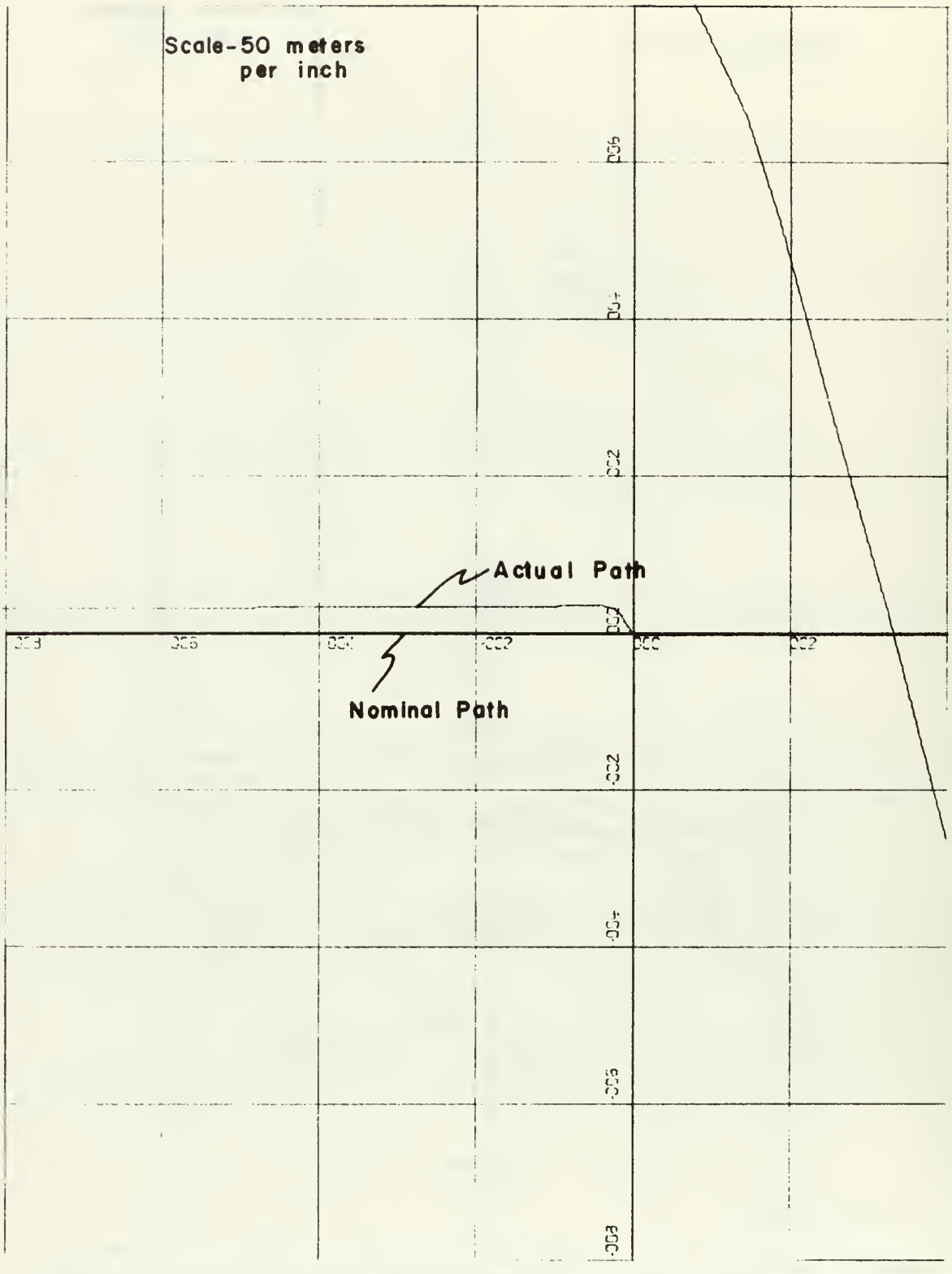


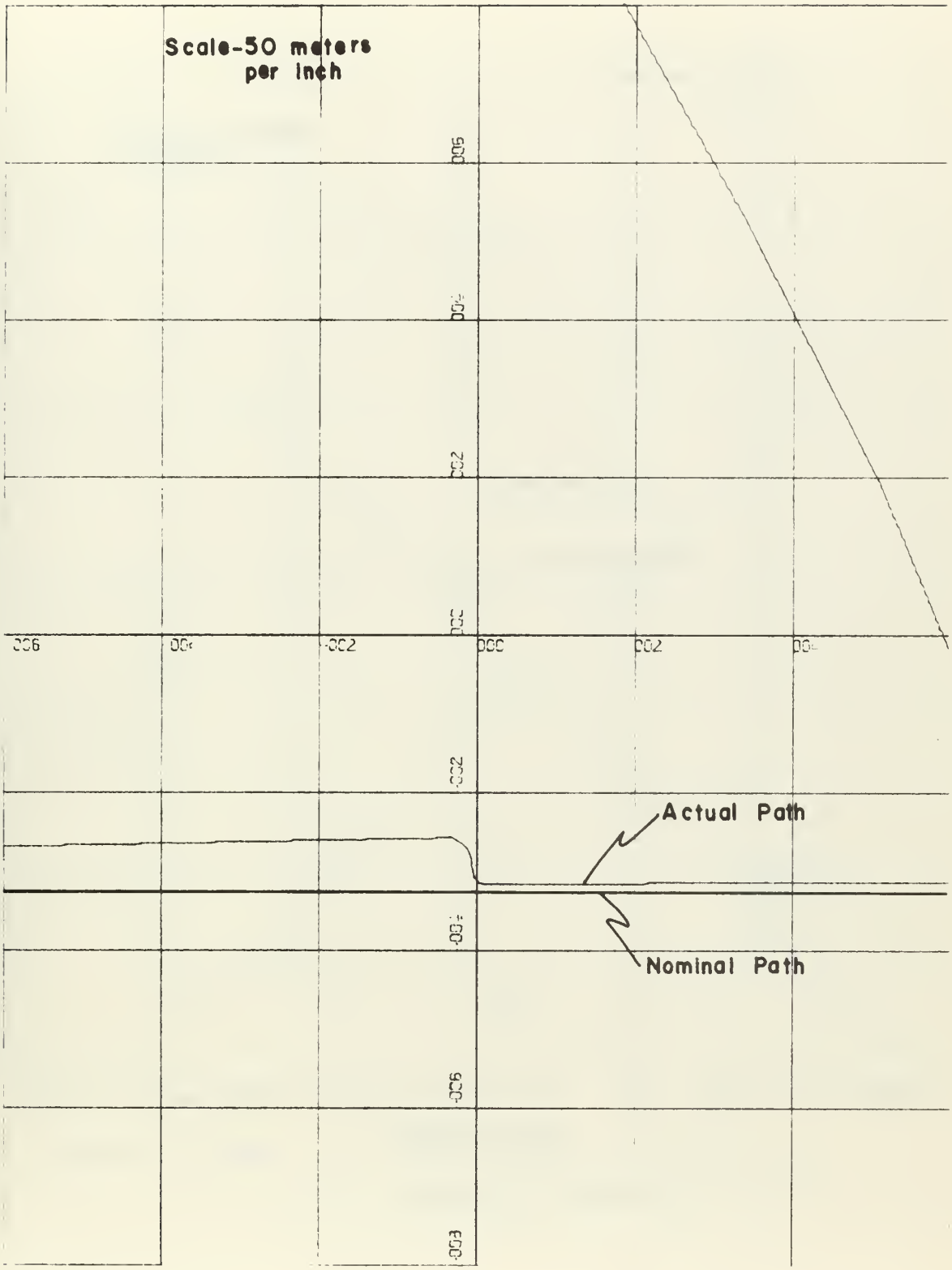
Fig. 3.13 Simulation six with the obstacles.



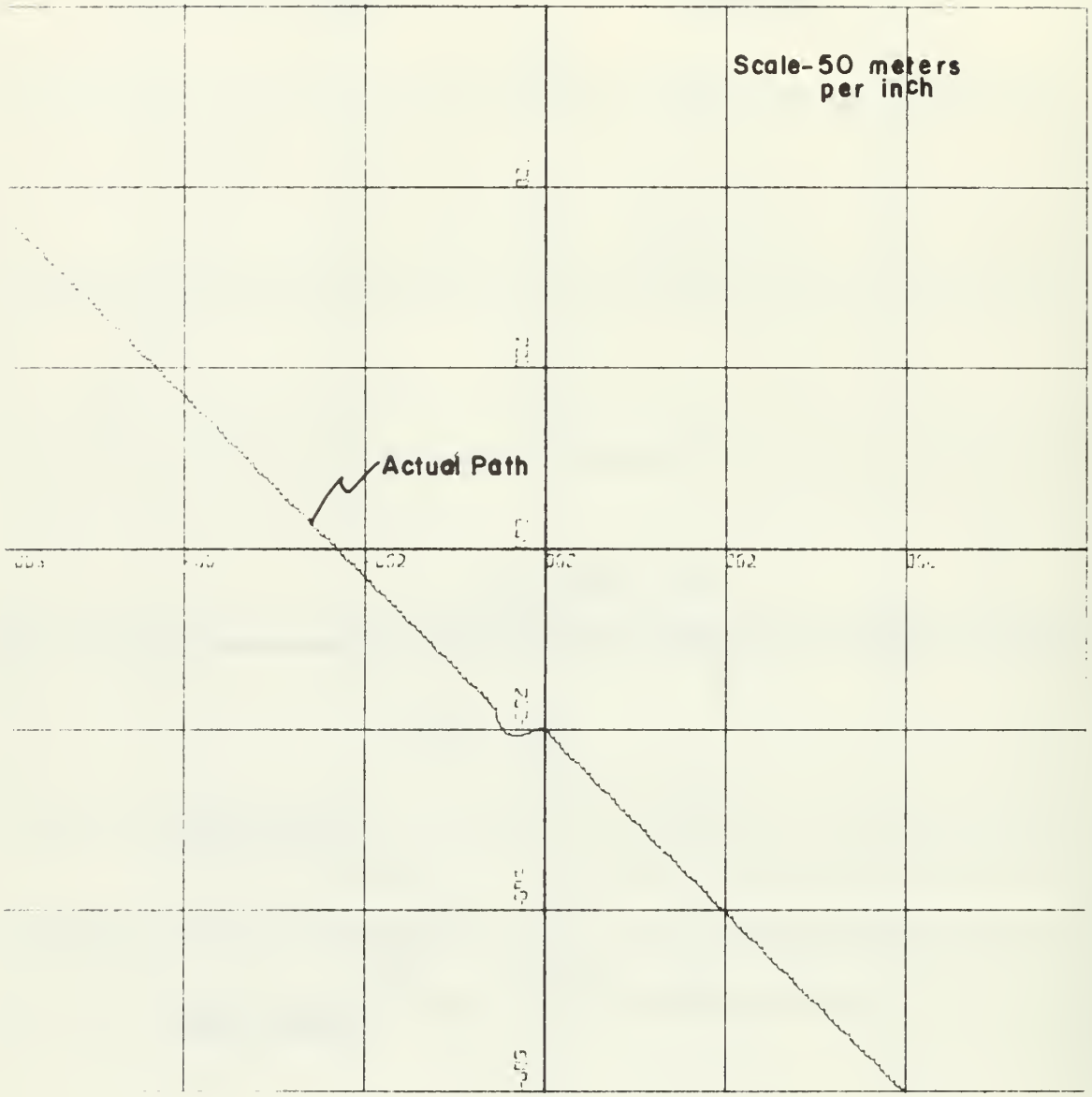
Insert A of Fig. 3.13 Illustrating the path taken around an obstacle.



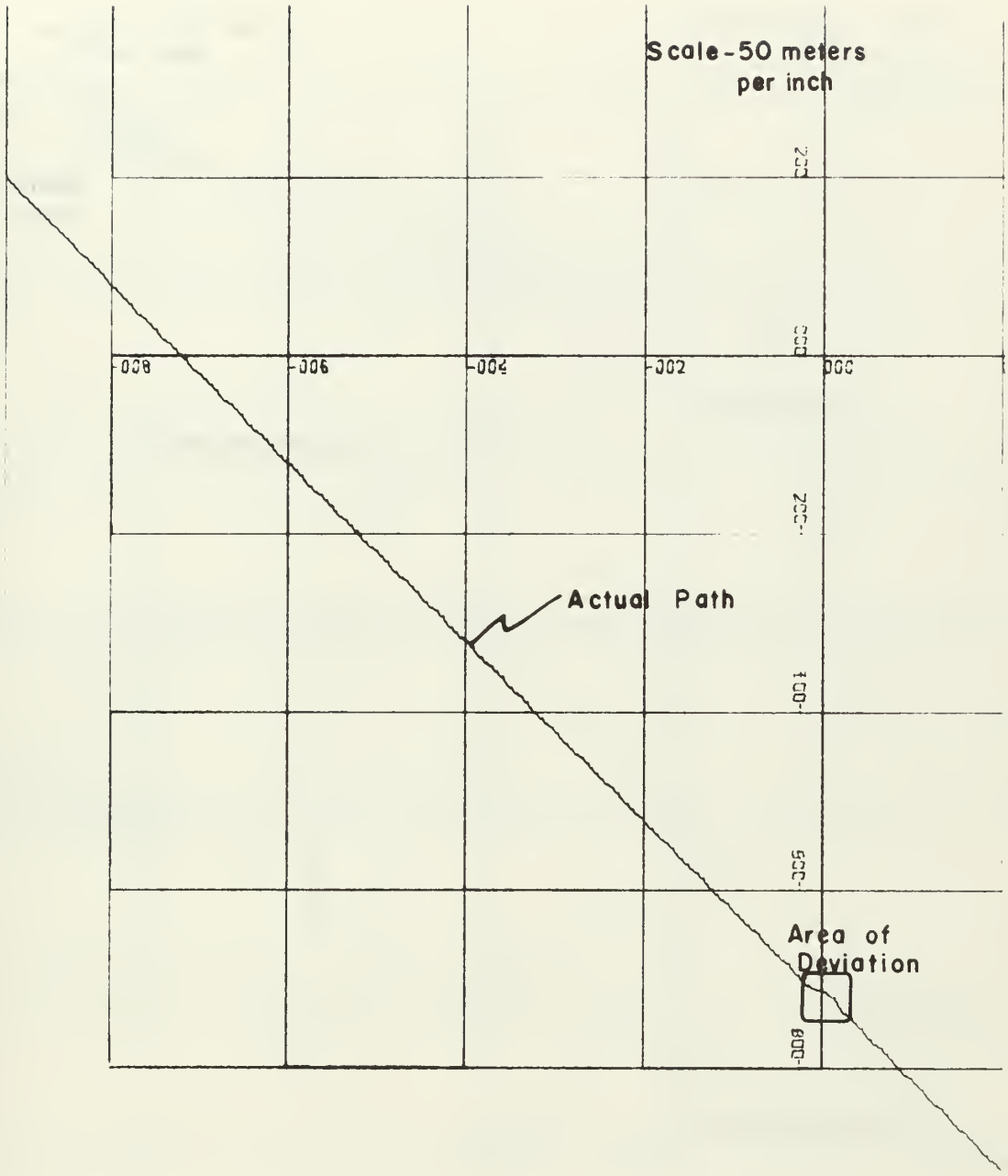
Insert B of Fig. 3.13 Illustrating the path taken around an obstacle.



Insert C of Fig. 3.13 Illustrating the path taken around an obstacle.



Insert D of Fig. 3.13 Illustrating the path taken around an obstacle.



Insert E of Fig. 3.13 Illustrating the path taken around an obstacle.

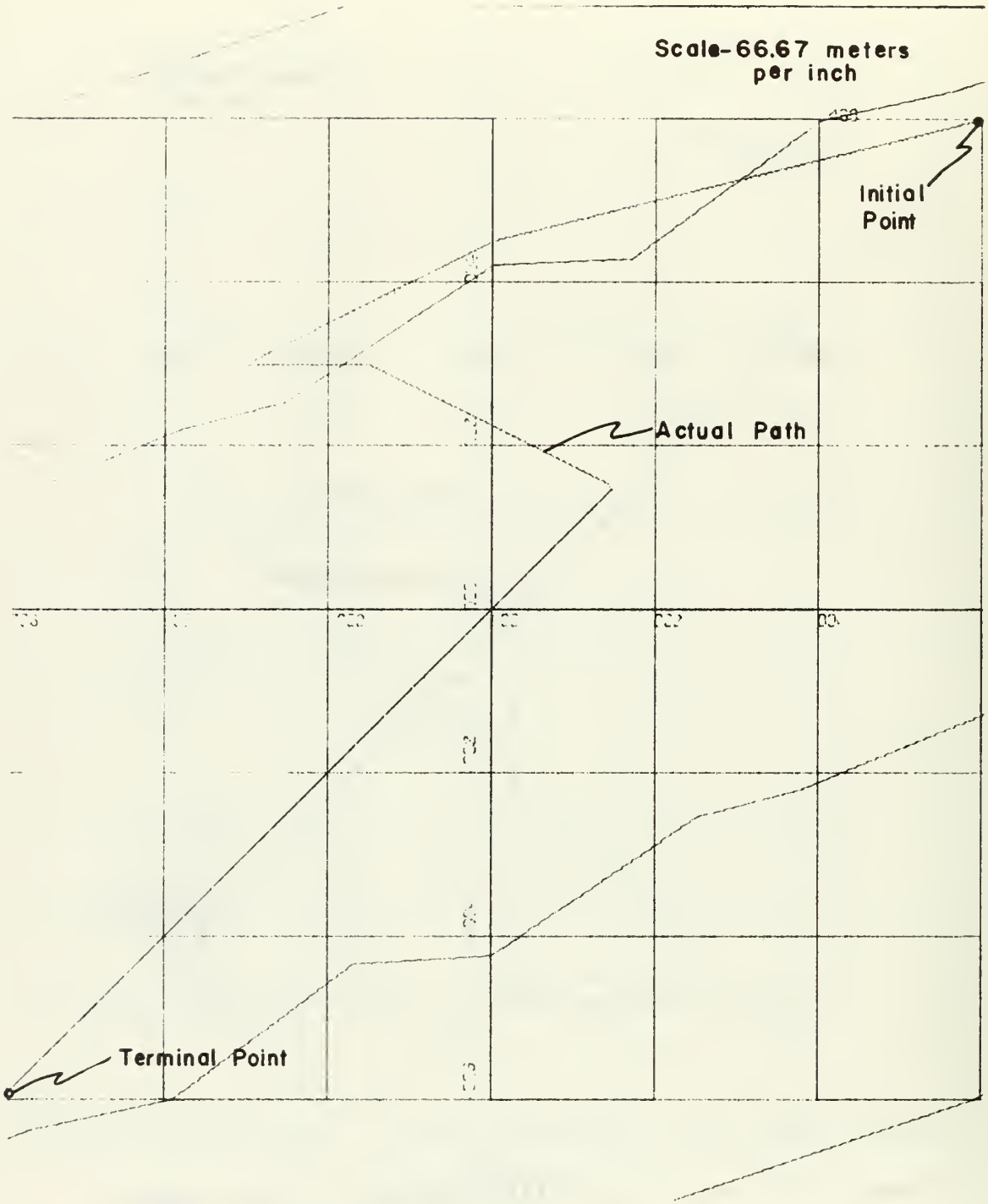


Fig. 3.14 Simulation seven.

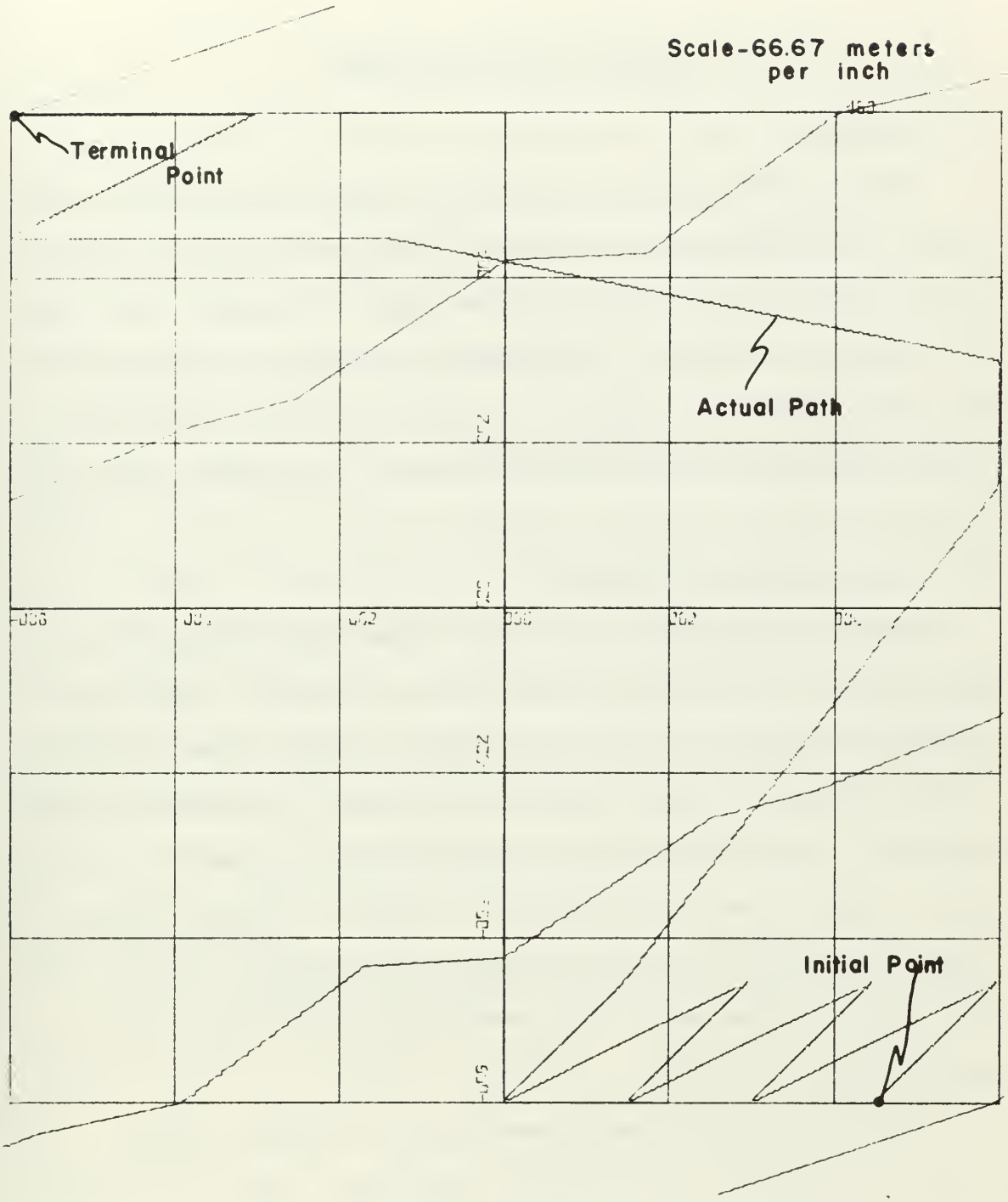


Fig. 3.15 Simulation eight.

III. SUMMARY AND CONCLUSIONS

A. SUMMARY

Chapter II reported on a survey that was undertaken to find a routing algorithm that was best suited for determining the optimal path over a large number of nodes. The survey showed that dynamic programming and Moore's algorithm D were best suited for use as the global routing algorithm. Dynamic programming was chosen because it was more adaptive to computer use and also a program by Dr. D. E. Kirk (Ref. K-2) was already available.

Moore's algorithms A,B, and C were much too restrictive because of the unity cost space; however, his algorithm D could prove to be very useful in the present application. Algorithm D is similar to dynamic programming, but because of the difficulty in adapting it to computer use it was not used. Lee's algorithm is similar to Moore's algorithm A, but even the added feature of including the second cost factor proved to be too restrictive for the present application.

Dantzig's algorithms proved to be of little value because of the many calculations required. Both algorithms were better than direct enumeration only because they allowed the topography of the map to eliminate those points that were not connected.

In Chapter III, Lim's path-finding algorithm was introduced. In order to show the usefulness of Lim's algorithm a suitable terrain simulation was developed. With the use of two different shapes of hills a variety of terrain features could be simulated. The disposition of the debris around the hills was successful in illustrating the performance of Lim's algorithm. Just how realistic the simulation of the debris is cannot be seen at the present time because no comparisons with actual data have been made.

Because of the elevation limitation Lim's original algorithm restricted the vehicle to too small an area of exploration. After changing the elevation limit to a slope limitation, the vehicle was given greater versatility and maneuverability. The other problems associated with the change to a slope limitation were solved.

B. CONCLUSIONS

The combination of dynamic programming and Lim's algorithm provides an efficient method of maneuvering an unmanned robot over terrain whose topographical features are only partially known. The simulations illustrated this fact satisfactorily. It is concluded that except for some minor difficulties, the algorithms are well suited for use with an unmanned robot.

The combined algorithm may be improved by making it an adaptive learning process by recording significant data,

e.g., true length/expected length, or mean path length between obstacles. At the end of each exploration this data could be used to re-calculate the optimal paths and to up-date the cost figures for all types of terrains that are of a similar nature.

LIST OF REFERENCES

B-1 Bellman, R. E., and Dreyfus, S.E., Applied Dynamic Programming, Princeton University Press, 1957.

B-2 Bellman, R. E., Dynamic Programming, Princeton University Press, 1957.

B-3 Bellman, R. E.,and Kalba, R. E., Dynamic Programming and Modern Control Theory, Academic Press, 1965.

D-1 Dantzig, G. B., "Discrete-Variable Extremum Problems", Operations Research, v.5,No.2, p.266-277, 1959.

D-2 Dayhoff, M. O., "A Contour-Map Program for X-Ray Crystallography", Communications of Association for Computing Machinery, October 1963.

K-1 Kirk, D. E., Optimal Control Theory: An Introduction, Prentice-Hall, Inc., to be published 1970.

K-2 Jet Propulsion Laboratory Technical Report 32-1369, A Path-Finding Algorithm for an Unmanned Roving Vehicle, by D. E. Kirk, 15 May 1969.

L-1 Lee, C. Y., "An Algorithm for Path Connections and Its Applications", IEEE Transactions Electronic Computers, p.346-365, September 1961.

L-2 Jet Propulsion Laboratory Technical Report 32-1288, A Path-Finding Algorithm for a Myopic Robot, by L. Y. Lim, 1 August 1968.

M-1 Moore, E. F., "Shortest Path through a Maze", Annals of the Computation Laboratory of Harvard University, v.30, p.285-292, 1959.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	20
2.	Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3.	Commandant of the Marine Corps (Code AO3C) Headquarters, U.S. Marine Corps Washington, D. C. 20380	1
4.	James Carson Breckenridge Library Marine Corps Development and Educational Command Quantico, Virginia 22134	1
5.	Associate Professor Donald E. Kirk Department of Electrical Engineering Naval Postgraduate School Monterey, California 93940	2
6.	2/LT Marvin H. Floom, Jr. USMC 14677 Saltamontes Way Los Altos, California 94022	2
7.	Mr. Larry Y. Lim The Jet Propulsion Laboratory 4800 Oak Grove Drive Pasadena, California 91103	1

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1 ORIGINATING ACTIVITY *(Corporate author)*

Naval Postgraduate School
Monterey, California 93940

2a. REPORT SECURITY CLASSIFICATION

Unclassified

2b. GROUP

3 REPORT TITLE

The Combination of a Global Routing Algorithm and a Path-Finding Algorithm for an Unmanned Roving Vehicle

4 DESCRIPTIVE NOTES *(Type of report and inclusive dates)*

Master's Thesis; October 1969

5 AUTHOR(S) *(First name, middle initial, last name)*

Marvin H. Floom, Jr.

6. REPORT DATE

October 1969

7a. TOTAL NO. OF PAGES

101

7b. NO. OF REFS

10

8a. CONTRACT OR GRANT NO.

b. PROJECT NO.

c.

d.

9a. ORIGINATOR'S REPORT NUMBER(S)

9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*

10. DISTRIBUTION STATEMENT

This document has been approved for public release and sale; its distribution is unlimited.

11. SUPPLEMENTARY NOTES

12. SPONSORING MILITARY ACTIVITY

Naval Postgraduate School
Monterey, California 93940

13. ABSTRACT

In future space missions it is planned that an unmanned robot will be sent to explore the other planets' surface. Control of the vehicle from earth is unrealistic because of the long delay time in the transmission of data. From a gross knowledge of the terrain a global routing algorithm can be used to find an optimal path from one point to another. A survey was undertaken to find an algorithm best suited for this use. Dynamic programming was selected and in combination with Lim's path-finding algorithm proved to be successful in simulated vehicle explorations over terrain represented by Gaussian density functions.

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	W
Unmanned Robot						
Path-Finding Algorithm						
Global Routing Algorithm						